

Trabajo Django

Francisco Javier del Valle Rodríguez

Índice

Objetivos	4
1. Instalación	5
2. Empezar proyecto nuevo	8
3. Primera migración	10
4. Creación de Superusuarios y Administración de Django.	11
5. Primera aplicación	15
6. Primer modelo	16
7. Crear objetos y registrar modelo en administración	18
8. Personalizar modelo en el admin	21
9. Primera vista	22
10. Configuración de plantillas	24
11. Escribir Formulario.	25
12. Formulario en una vista	25
13. Método HTTP POST en formulario	26
14. Validaciones formulario pt.1	27
15. Guardar datos del formulario con el modelo.	29
16. Model Form	31
17. Validaciones Model Form	33
18. Contexto en la vista, plantillas	36
19. Model Form en la vista	39
20. Custom Form para contacto	45
21. Configurar email.	52
22. Configuración de archivos estáticos	53
23. Configuración Bootstrap	55
24. Plantillas	59
25. Django Crispy Forms	65
26. Estilo: Bootstrap 1	67
27. Estilo: CSS Custom	72

28. Enlaces con nombres URL	74
29. Estilo: Bootstrap 2	76
30. Django Registration Redux	77
31. Django Registration Redux 2	81
32. Cambiar URL Redirect después de login	85
33. Autenticación para enlaces en la Navbar	86
34. Formulario de login en la Navbar.	88

Objetivos

Crear un documento donde se vea un seguimiento del curso gratuito de Udemy sobre Django, donde se recoja el proyecto, los problemas que tengamos y cómo solucionar los fallos que encontremos en el proyecto

¿Qué es PIP ?

PIP (Pip Instalador de Paquetes o Pip Instalador de Python) es un sistema de gestión de paquetes utilizado para instalar y administrar paquetes de software escritos en Python.

¿Qué es Python ?

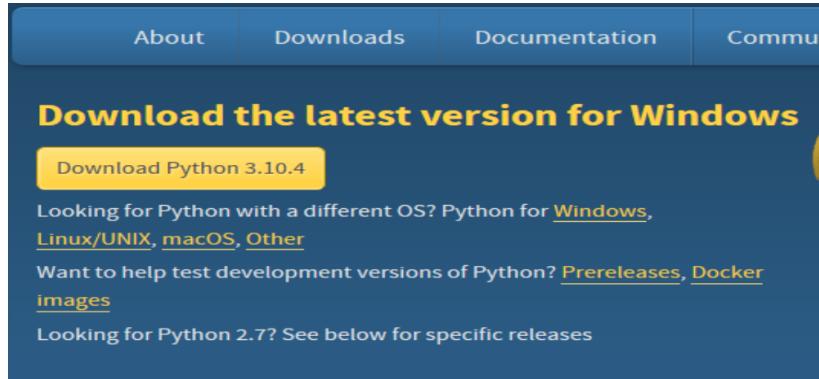
Es un lenguaje de programación interpretado, cuya filosofía hace hincapié en la legibilidad de su código y la orientación a objetos. Es un lenguaje interpretado, dinámico y multiplataforma, administrado por la Python Software Foundation.

¿Qué es Django ?

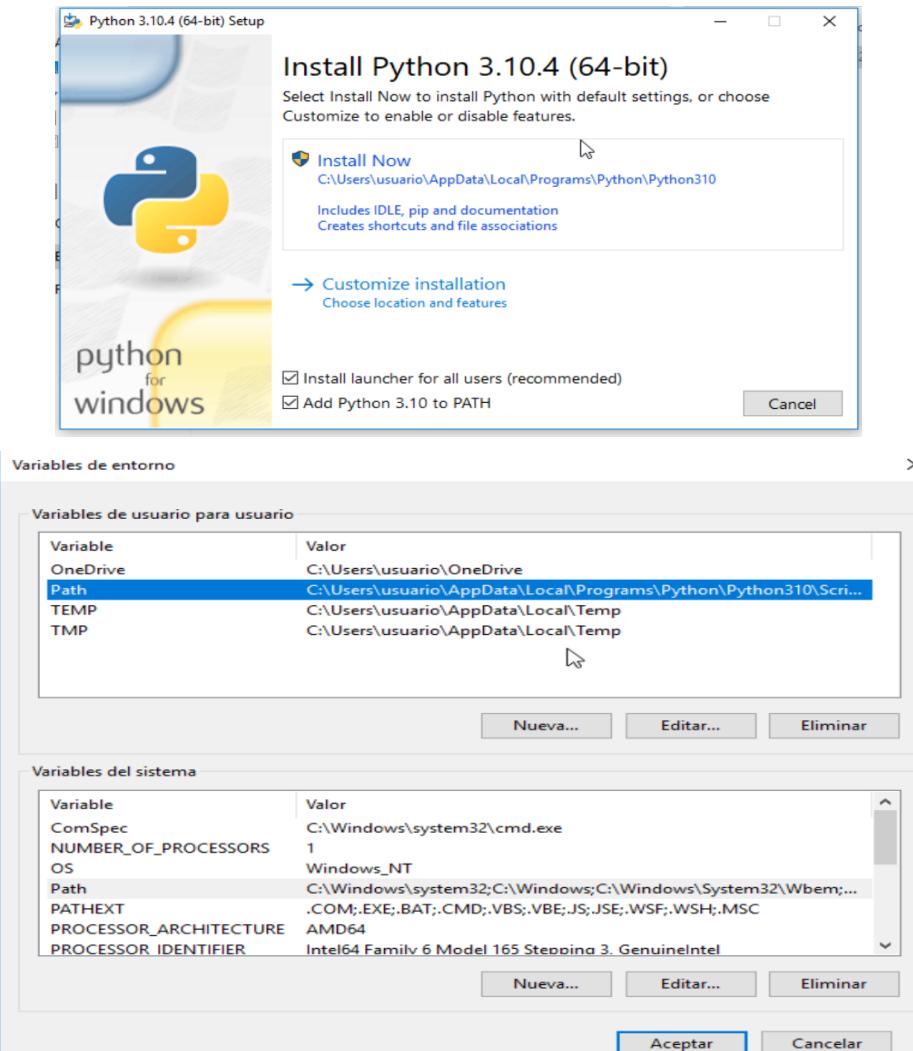
Es un framework (o entorno de trabajo) de desarrollo web de código abierto, escrito en Python, que respeta el patrón de diseño conocido como modelo–vista–controlador (MVC), un patrón de arquitectura de software, que separa los datos y principalmente lo que es la lógica de negocio de una aplicación de su representación y el módulo encargado de gestionar los eventos y las comunicaciones. La meta fundamental de Django es facilitar la creación de sitios web complejos.

1. Instalación

Descargamos la versión más moderna de python en este caso python 3.10.4 que la podremos descargar en el siguiente enlace <https://www.python.org/downloads/>



Al seleccionar la opción “Add Python 3.10 to PATH” se nos creará la variable de entorno



Nos vamos a símbolo del sistema y ejecutamos python para ver si la instalación se ha hecho correctamente

```
C:\Users\usuario>python
Python 3.10.4 (tags/v3.10.4:9d38120, Mar 23 2022, 23:13:41) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 2*3
6
>>> nombre = input('Como te llamas?')
Como te llamas?Fran
>>> print('Hola, %s.' %nombre)
Hola, Fran.
>>>
```

En esta versión de python ya viene instalado pip por lo que el siguiente paso es instalar Virtual Environment. Para ello, ejecutamos el siguiente comando “pip install virtualenv”

```
C:\Users\usuario>python -m pip install -U pip
Requirement already satisfied: pip in c:\users\usuario\appdata\local\programs\python\python310\lib\site-packages (22.0.4)

C:\Users\usuario>pip install virtualenv
Collecting virtualenv
  Downloading virtualenv-20.14.1-py2.py3-none-any.whl (8.8 MB)
    ..... 8.8/8.8 MB 3.4 MB/s eta 0:00:00
Collecting distlib<1,>=0.3.1
  Downloading distlib-0.3.4-py2.py3-none-any.whl (461 kB)
    ..... 461.2/461.2 KB 4.1 MB/s eta 0:00:00
Collecting filelock<4,>=3.2
  Downloading filelock-3.6.0-py3-none-any.whl (10.0 kB)
Collecting platformdirs<3,>=2
  Downloading platformdirs-2.5.2-py3-none-any.whl (14 kB)
Collecting six<2,>=1.9.0
  Downloading six-1.16.0-py2.py3-none-any.whl (11 kB)
Installing collected packages: distlib, six, platformdirs, filelock, virtualenv
Successfully installed distlib-0.3.4 filelock-3.6.0 platformdirs-2.5.2 six-1.16.0 virtualenv-20.14.1

C:\Users\usuario>
```

Instalamos Virtual Environment con la finalidad de poder instalar Django dentro de él, para ello creamos uno usando el comando “virtualenv test_env”, no metemos dentro y lo activamos ejecutando “.\Scripts\activate”

```
C:\Users\usuario>virtualenv test_env
created virtual environment CPython3.10.4.final.0-64 in 11531ms
  creator CPython3Windows(dest=C:\Users\usuario\test_env, clear=False, no_vcs_ignore=False, global=False)
  seeder FromAppData(download=False, pip=bundle, setuptools=bundle, wheel=bundle, via=copy, app_data_dir=C:\Users\usuario\AppData\Local\pypa\virtualenv)
    added seed packages: pip==22.0.4, setuptools==62.1.0, wheel==0.37.1
  activators BashActivator,BatchActivator,FishActivator,NushellActivator,PowerShellActivator,PythonActivator

C:\Users\usuario>cd test_env

C:\Users\usuario\test_env>.\Scripts\activate
(test_env) C:\Users\usuario\test_env>
```

Una vez hecho esto, ya podremos instalar Django, para ello utilizaremos el comando “pip install django”

```
(test_env) C:\Users\usuario\test_env> pip install django
Collecting django
  Downloading Django-4.0.4-py3-none-any.whl (8.0 MB)
    8.0/8.0 MB 3.2 MB/s eta 0:00:00
Collecting tzdata
  Downloading tzdata-2022.1-py2.py3-none-any.whl (339 kB)
    339.5/339.5 KB 5.2 MB/s eta 0:00:00
Collecting sqlparse>=0.2.2
  Downloading sqlparse-0.4.2-py3-none-any.whl (42 kB)
    42.3/42.3 KB ? eta 0:00:00
Collecting asgiref<4,>=3.4.1
  Downloading asgiref-3.5.0-py3-none-any.whl (22 kB)
Installing collected packages: tzdata, sqlparse, asgiref, django
Successfully installed asgiref-3.5.0 django-4.0.4 sqlparse-0.4.2 tzdata-2022.1

(test env) C:\Users\usuario\test env>
```

Vamos a crear un proyecto nuevo con Django, vamos a escribir lo siguiente ‘python .\Scripts\django-admin.py startproject test_project’. Con esto indicamos que queremos iniciar un nuevo proyecto con su nombre. Para asegurarnos de su correcta instalación, podemos hacer un listado del contenido de nuestro Virtual Environment.

```
(test_env) C:\Users\usuario\test_env>python .\Scripts\django-admin.exe startproject test_project

(test_env) C:\Users\usuario\test_env>dir
El volumen de la unidad C no tiene etiqueta.
El n mero de serie del volumen es: 2222-5B40

Directorio de C:\Users\usuario\test_env

18/04/2022 18:00 <DIR> .
18/04/2022 18:00 <DIR> ..
18/04/2022 17:40 42 .gitignore
18/04/2022 17:40 <DIR> Lib
18/04/2022 17:40 418 pyenv.cfg
18/04/2022 17:54 <DIR> Scripts
18/04/2022 18:00 <DIR> test_project
      2 archivos        460 bytes
      5 dirs  39.392.100.352 bytes libres

(test_env) C:\Users\usuario\test_env>
```

```
(test_env) C:\Users\usuario\test_env>cd test_project

(test_env) C:\Users\usuario\test_env\test_project>dir
El volumen de la unidad C no tiene etiqueta.
El n mero de serie del volumen es: 2222-5B40

Directorio de C:\Users\usuario\test_env\test_project

18/04/2022 18:00 <DIR> .
18/04/2022 18:00 <DIR> ..
18/04/2022 18:00       690 manage.py
18/04/2022 18:00 <DIR> test_project
      1 archivos        690 bytes
      3 dirs  39.392.030.720 bytes libres

(test_env) C:\Users\usuario\test_env\test_project>
```

2. Empezar proyecto nuevo

Para empezar un proyecto nuevo, abrimos nuestro cmd y creamos un nuevo directorio (en mi caso voy a crear una carpeta llamada “proyecto” en el escritorio). Entramos en ella e instalamos el Virtual Environment. Después, lo activamos como hicimos en el punto anterior.

```
C:\Users\usuario>cd Desktop  
C:\Users\usuario\Desktop>mkdir proyecto && cd proyecto  
  
C:\Users\usuario\Desktop\proyecto>virtualenv .  
created virtual environment CPython3.10.4.final.0-64 in 3499ms  
  creator CPython3Windows(dest=C:\Users\usuario\Desktop\proyecto, clear=False, no_vcs_ignore=False, global=False)  
  seeder FromAppData(download=False, pip=bundle, setuptools=bundle, wheel=bundle, via=copy, app_data_dir=C:\Users\usuario\AppData\Local\pypa\virtualenv)  
  added seed packages: pip==22.0.4, setuptools==62.1.0, wheel==0.37.1  
  activators BashActivator,BatchActivator,FishActivator,NushellActivator,PowerShellActivator,PythonActivator  
  
C:\Users\usuario\Desktop\proyecto>.\Scripts\activate
```

A continuación instalaremos Django, utilizamos el comando “pip install django”

```
(proyecto) C:\Users\usuario\Desktop\proyecto>pip install django  
Collecting django  
  Using cached Django-4.0.4-py3-none-any.whl (8.0 MB)  
Collecting asgiref<4,>=3.4.1  
  Using cached asgiref-3.5.0-py3-none-any.whl (22 kB)  
Collecting tzdata  
  Using cached tzdata-2022.1-py2.py3-none-any.whl (339 kB)  
Collecting sqlparse>=0.2.2  
  Using cached sqlparse-0.4.2-py3-none-any.whl (42 kB)  
Installing collected packages: tzdata, sqlparse, asgiref, django  
Successfully installed asgiref-3.5.0 django-4.0.4 sqlparse-0.4.2 tzdata-2022.1  
  
(proyecto) C:\Users\usuario\Desktop\proyecto>
```

Creamos el nuevo proyecto para ello usamos “.\Scripts\django-admin.py startproject proyecto”

```
(proyecto) C:\Users\usuario\Desktop\proyecto>python .\Scripts\django-admin.exe startproject proyecto  
(proyecto) C:\Users\usuario\Desktop\proyecto>
```

Una vez creado el proyecto nos iremos a Escritorio y dentro de la carpeta proyecto cambiaremos el nombre de la carpeta proyecto por src para no crear confusiones.

Carpeta proyecto		Nombre	Fecha de modificación	Tipo	Tamaño
id	o	Lib	18/04/2022 18:15	Carpeta de archivos	
s	tos	src	18/04/2022 18:21	Carpeta de archivos	
n	tos	Scripts	18/04/2022 18:16	Carpeta de archivos	
;	o	.gitignore	18/04/2022 18:15	Archivo GITIGNORE	1 KB
;	o	pyvenv.cfg	18/04/2022 18:15	Archivo CFG	1 KB

Cambiamos al directorio src y ejecutamos el servidor de desarrollo de nuestro entorno de desarrollo, para ello utilizamos el comando “python manage.py runserver”

```
( proyecto ) C:\Users\usuario\Desktop\proyecto>cd src  
  
( proyecto ) C:\Users\usuario\Desktop\proyecto\src>python manage.py runserver  
Watching for file changes with StatReloader  
Performing system checks...  
  
System check identified no issues (0 silenced).  
  
You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin,  
auth, contenttypes, sessions.  
Run 'python manage.py migrate' to apply them.  
April 18, 2022 - 18:31:06  
Django version 4.0.4, using settings 'proyecto.settings'  
Starting development server at http://127.0.0.1:8000/  
Quit the server with CTRL-BREAK.
```

A continuación en el navegador ponemos “http://127.0.0.1:8000”, y podemos comprobar que funciona correctamente.

 django
View [release notes](#) for Django 4.0



The install worked successfully! Congratulations!

3. Primera migración

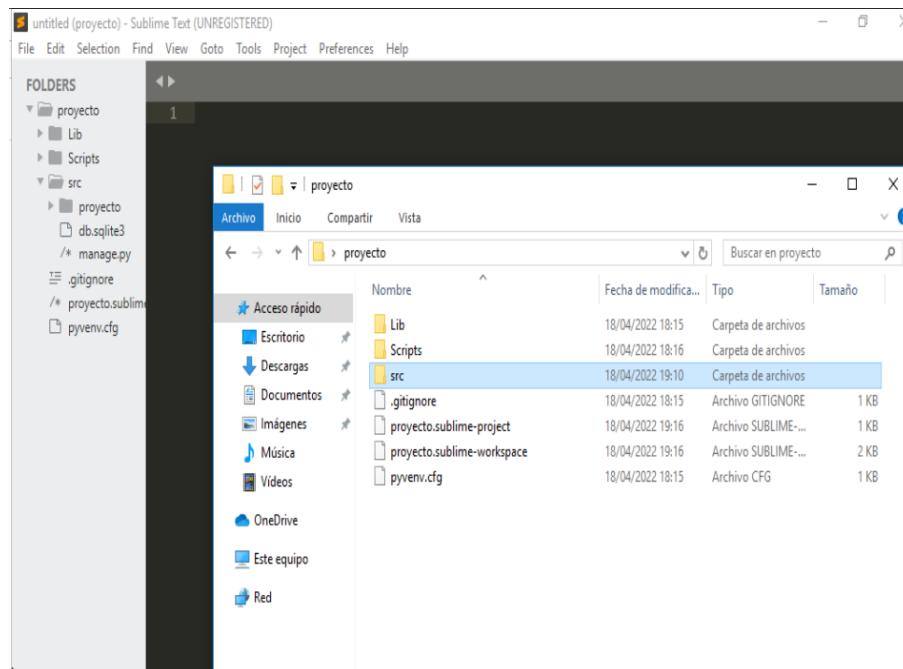
Ya tenemos nuestro proyecto funcionando. Ahora podemos hacer las migraciones. Las migraciones nos sirven para comunicarnos con nuestra base de datos, es el vínculo entre la base de datos y nuestro proyecto.

En nuestro cmd, cerramos el servidor (Ctrl + D) y escribimos 'python manage.py migrate'.

```
(proyecto) C:\Users\usuario\Desktop\proyecto\src>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running Migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying sessions.0001_initial... OK

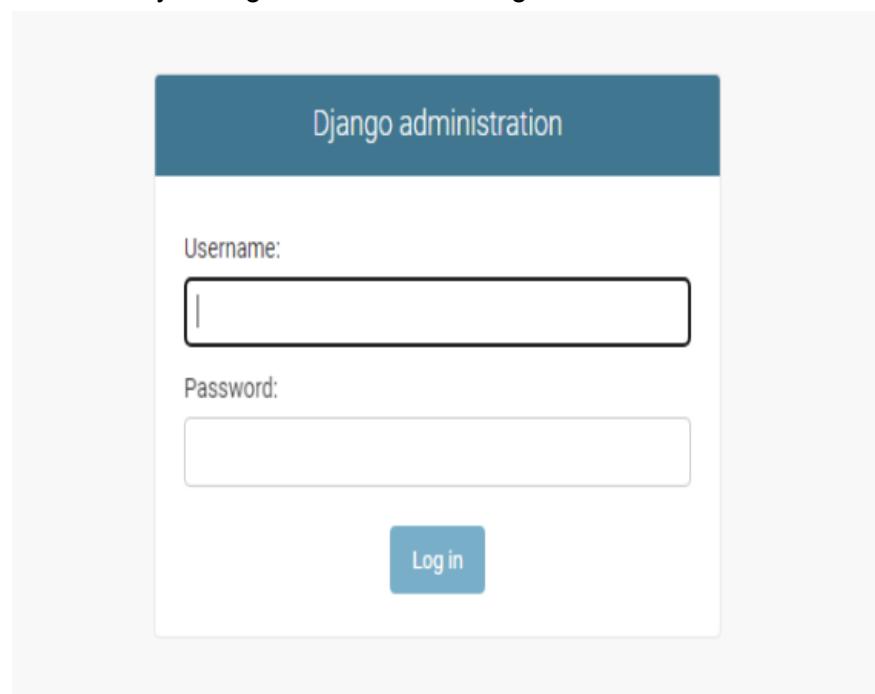
(proyecto) C:\Users\usuario\Desktop\proyecto\src>
```

A continuación, abrimos nuestro editor de texto (en mi caso Sublime Text) y vamos a trabajar con nuestro proyecto. Pinchamos en 'Project' y añadimos la carpeta del proyecto 'proyecto'. Una vez abierto, lo guardamos dentro de nuestra carpeta 'raíz'.



4. Creación de Superusuarios y Administración de Django.

Los proyectos de Django vienen automáticamente con una interfaz ya escrita. Volvemos a activar nuestro servidor y recargamos nuestro navegador web.



Para poder iniciar sesión dentro de Django, cerramos el servidor y vamos a crear un superusuario, capaz de entrar. Para ello, ponemos en nuestro cmd 'python manage.py createsuperuser' y añadimos nuestro usuario, email y contraseña.

```
( proyecto) C:\Users\usuario\Desktop\proyecto\src>python manage.py createsuperuser
Username (leave blank to use 'usuario'): Fran
Email address: franciscojavier.delvalle.rodriguez.alu@iesfernandoaguilar.es
Password:
Password (again):
This password is too short. It must contain at least 8 characters.
Bypass password validation and create user anyway? [y/N]: N
Password:
Password (again):
Error: Your passwords didn't match.
Password:
Password (again):
Superuser created successfully.

( proyecto) C:\Users\usuario\Desktop\proyecto\src>
```

Volvemos a iniciar el servidor y ya podemos acceder a nuestra página. Esta es la página principal de la interfaz administrativa.

The screenshot shows the Django administration interface at the URL `127.0.0.1:8000/admin/`. The top navigation bar includes links for 'WELCOME, FRAN', 'VIEW SITE / CHANGE PASSWORD / LOG OUT'. Below the header, the main content area is titled 'Site administration' and features a 'AUTHENTICATION AND AUTHORIZATION' section. This section contains two tabs: 'Groups' and 'Users'. Under 'Groups', there is a '+ Add' button and a 'Change' link. Under 'Users', there is also a '+ Add' button and a 'Change' link. To the right of the main content, there are two sidebar boxes: 'Recent actions' (empty) and 'My actions' (also empty, with the message 'None available').

Cambiamos el idioma para ello en el fichero settings.py modificaremos la línea que ponga "LANGUAGE_CODE = 'en-us'" por LANGUAGE_CODE= 'es'" y recargamos la página.

The screenshot shows a code editor with a file named 'settings.py' open. On the left, a file tree displays the project structure: 'FOLDERS' include 'proyecto', 'Lib', 'Scripts', and 'src'. 'src' contains 'proyecto', '_pycache_/_init_.py', 'asgi.py', and 'settings.py'. The 'settings.py' file is selected. The code editor shows the following content:

```
94     {
95         'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidation',
96     },
97     [
98         {
99             'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidation',
100        }
101    ],
102
103 # Internationalization
104 # https://docs.djangoproject.com/en/4.0/topics/i18n/
105
106 LANGUAGE_CODE = 'es'
107
108 TIME_ZONE = 'UTC'
109
110 USE_I18N = True
111
112 USE_TZ = True
113
114
115 # Static files (CSS, JavaScript, Images)
116 # https://docs.djangoproject.com/en/4.0/howto/static-files/
117
118 STATIC_URL = 'static/'
119
120 # Default primary key field type
121 # https://docs.djangoproject.com/en/4.0/ref/settings/#default-auto-field
122
123 DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
124
```

Administración de Django

BIENVENIDOS, FRAN. [VER EL SITIO](#) / [CAMBIAR CONTRASEÑA](#) / [CERRAR SESIÓN](#)

Sitio administrativo

AUTENTICACIÓN Y AUTORIZACIÓN

Grupos Modelos en la aplicación Autenticación y autorización

Usuarios

Añadir Modificar

Añadir Modificar

Acciones recientes

Mis acciones

Ninguno disponible

Si, por ejemplo, queremos crear otro usuario, podemos entrar en la sección de ‘Usuarios’ y ‘Añadir usuario’

Empiece a escribir para filtrar...

AUTENTICACIÓN Y AUTORIZACIÓN

Grupos + Añadir

Usuarios + Añadir

Añadir usuario

Primero, ingrese un nombre de usuario y contraseña. Luego, podrá editar más opciones del usuario.

Nombre de usuario: Requerido. 150 caracteres como máximo. Únicamente letras, dígitos y @/./+/-/_

Contraseña:

Su contraseña no puede asemejarse tanto a su otra información personal.
Su contraseña debe contener al menos 8 caracteres.
Su contraseña no puede ser una clave utilizada comúnmente.
Su contraseña no puede ser completamente numérica.

Contraseña (confirmación):

Para verificar, introduzca la misma contraseña anterior.

Guardar y añadir otro Guardar y continuar editando GUARDAR

Si volvemos a ‘Usuarios’ podremos ver que ya tenemos el usuario creado. El cual no tendrá todos los permisos como nuestro superusuario.

Empiece a escribir para filtrar...

AUTENTICACIÓN Y AUTORIZACIÓN

Grupos	 Añadir
Usuarios	 Añadir

Seleccione usuario a modificar

Q |

Acción: seleccionados 0 de 2

<input type="checkbox"/>	NOMBRE DE USUARIO	DIRECCIÓN DE CORREO ELECTRÓNICO
<input type="checkbox"/>	Fran	franciscojavier.delvalle.rodriguez
<input type="checkbox"/>	usuario	

2 usuarios

5. Primera aplicación

Para crear una aplicación tendremos que cerrar el servidor y ejecutamos el siguiente comando “python manage.py startapp boletin”, y vemos que se nos a creado la aplicación

The screenshot shows a Sublime Text window with two panes. The left pane is a file browser with the following structure:

- untitled (proyecto) - Sublime Text (U)
- File Edit Selection Find View
- FOLDERS
 - proyecto
 - Lib
 - proyecto
 - Scripts
 - src
 - boletin
 - proyecto
 - db.sqlite3
 - /* manage.py
- .gitignore
- /* proyecto.sublime-project
- pyvenv.cfg

The right pane is a terminal window titled "Símbolo del sistema" (Symbol of the system) showing Apache logs:

```
[20/Apr/2022 12:55:43] "GET /static/admin/css/base.css HTTP/1.1" 200 19513
[20/Apr/2022 12:55:44] "GET /static/admin/css/dashboard.css HTTP/1.1" 200 380
[20/Apr/2022 12:55:44] "GET /static/admin/css/responsive.css HTTP/1.1" 200 18575
[20/Apr/2022 12:55:44] "GET /static/admin/js/nav_sidebar.js HTTP/1.1" 200 3401
[20/Apr/2022 12:55:44] "GET /static/admin/css/fonts.css HTTP/1.1" 304 0
[20/Apr/2022 12:55:44] "GET /static/admin/fonts/Roboto-Light-webfont.woff HTTP/1.1" 304 0
[20/Apr/2022 12:55:44] "GET /static/admin/fonts/Roboto-Regular-webfont.woff HTTP/1.1" 304 0
[20/Apr/2022 12:55:44] "GET /static/admin/fonts/Roboto-Bold-webfont.woff HTTP/1.1" 304 0
[20/Apr/2022 12:55:44] "GET /static/admin/icon-addlink.svg HTTP/1.1" 200 331
[20/Apr/2022 12:55:44] "GET /static/admin/img/icon-changelink.svg HTTP/1.1" 200 380
[20/Apr/2022 12:55:44] "GET /admin/auth/user/ HTTP/1.1" 200 8572
[20/Apr/2022 12:55:44] "GET /admin/jsi18n/ HTTP/1.1" 200 7640
[20/Apr/2022 12:55:44] "GET /static/admin/css/changelists.css HTTP/1.1" 200 6932
[20/Apr/2022 12:55:44] "GET /static/admin/js/urlify.js HTTP/1.1" 200 7902
[20/Apr/2022 12:55:44] "GET /static/admin/js/actions.js HTTP/1.1" 200 7872
[20/Apr/2022 12:55:44] "GET /static/admin/js/vendor/jquery/jquery.js HTTP/1.1" 200 288580
[20/Apr/2022 12:55:44] "GET /static/admin/js/jquery.init.js HTTP/1.1" 200 347
[20/Apr/2022 12:55:44] "GET /static/admin/js/prepopulate.js HTTP/1.1" 200 1531
[20/Apr/2022 12:55:44] "GET /static/admin/js/vendor/xregexp/xregexp.js HTTP/1.1" 200 232381
[20/Apr/2022 12:55:44] "GET /static/admin/js/core.js HTTP/1.1" 200 5698
[20/Apr/2022 12:55:44] "GET /static/admin/img/search.svg HTTP/1.1" 200 458
[20/Apr/2022 12:55:55] "GET /static/admin/js/adminRelatedObjectLookups.js HTTP/1.1" 200 5984
[20/Apr/2022 12:55:55] "GET /static/admin/img/icon-yes.svg HTTP/1.1" 200 436
[20/Apr/2022 12:55:55] "GET /static/admin/img/tooltag-add.svg HTTP/1.1" 200 331
[20/Apr/2022 12:55:55] "GET /static/admin/img/icon-no.svg HTTP/1.1" 200 560
[20/Apr/2022 12:55:55] "GET /static/admin/img/sorting-icons.svg HTTP/1.1" 200 1097
```

C:\Users\usuario\Desktop\proyecto\src>python manage.py startapp boletin

C:\Users\usuario\Desktop\proyecto\src>

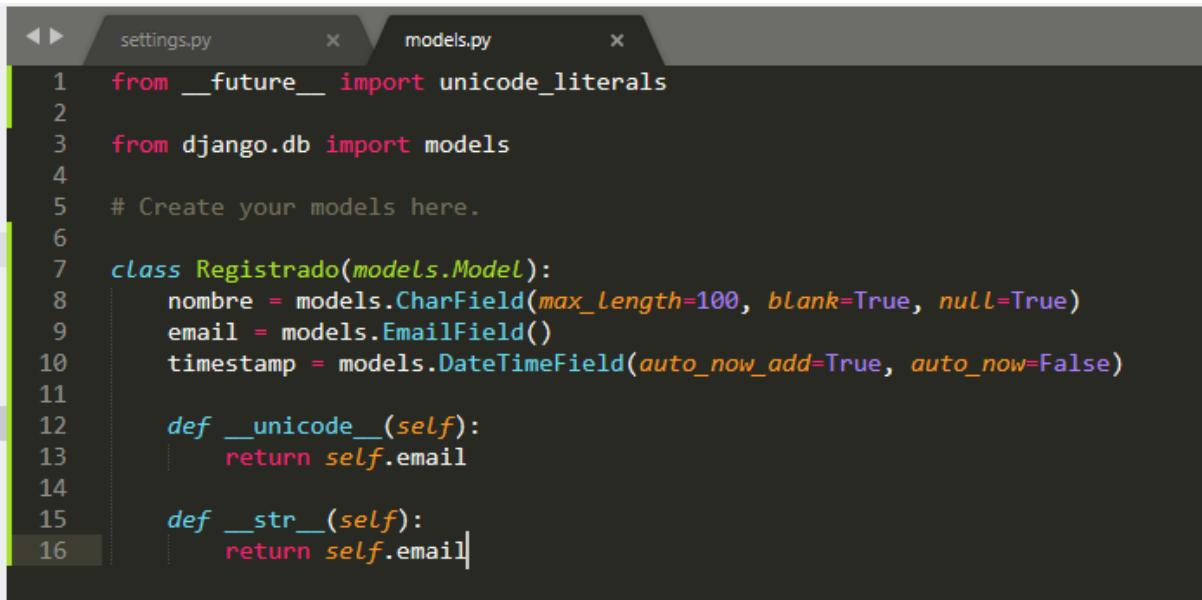
A continuación la registramos la aplicación en el fichero setting.py

The screenshot shows a code editor with the following Python code:

```
32
33 INSTALLED_APPS = [
34     'django.contrib.admin',
35     'django.contrib.auth',
36     'django.contrib.contenttypes',
37     'django.contrib.sessions',
38     'django.contrib.messages',
39     'django.contrib.staticfiles',
40     'boletin'
41 ]
```

6. Primer modelo

Vamos a crear un registro de la gente que va a recibir nuestro boletín. Para ello escribimos dentro de 'models.py' lo siguiente:



```
1 from __future__ import unicode_literals
2
3 from django.db import models
4
5 # Create your models here.
6
7 class Registrado(models.Model):
8     nombre = models.CharField(max_length=100, blank=True, null=True)
9     email = models.EmailField()
10    timestamp = models.DateTimeField(auto_now_add=True, auto_now=False)
11
12    def __unicode__(self):
13        return self.email
14
15    def __str__(self):
16        return self.email
```

Por último, realizamos la migración, para ello escribimos el siguiente comando "python manage.py makemigrations", que buscara todas las modificaciones que hemos hecho en nuestro modelo y va a empaquetarlas. Despues, con el comando "python manage.py migrate" se comunica con nuestra base de datos

```
C:\Users\usuario\Desktop\proyecto\src>python manage.py makemigrations
Migrations for 'boletin':
  boletin\migrations\0001_initial.py
    - Create model Registrado

C:\Users\usuario\Desktop\proyecto\src>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, boletin, contenttypes, sessions
Running migrations:
  Applying boletin.0001_initial... OK

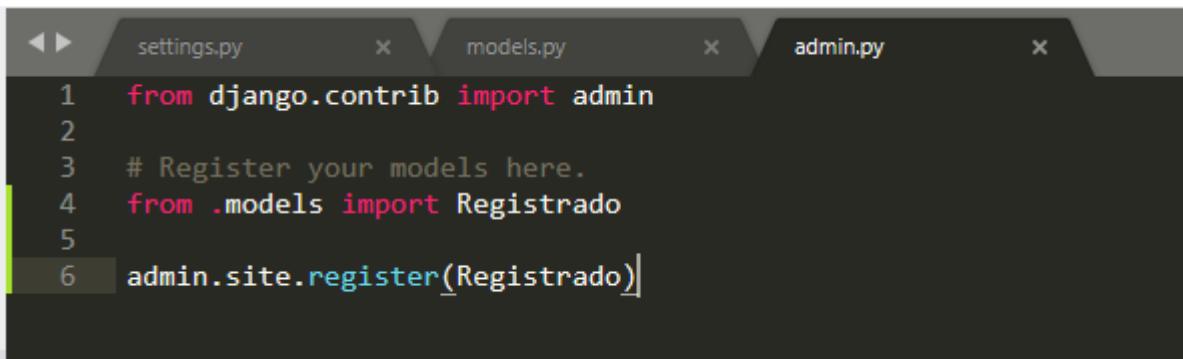
C:\Users\usuario\Desktop\proyecto\src>
```

7. Crear objetos y registrar modelo en administración

Ya podemos crear objetos y guardarlos en nuestra base de datos, en este caso usaremos el shell de Python. Para iniciar lo tendremos que usar el comando “python manage.py shell” e importaremos nuestro modelo, para ello utilizamos el siguiente comando “from boletin.models import Registrado”, y le añadiremos una variable, por ejemplo “gente = Registrado.objects.all()” la cual estará vacía, para guardar objetos podemos hacer “persona1 = Registrado.objects.create(nombre='Fran',email='franciscojavier.delvalle.rodriguez@iesfernandoagUILAR.es”)

```
C:\Users\usuario\Desktop\proyecto\src>python manage.py shell
Python 3.10.4 (tags/v3.10.4:9d38120, Mar 23 2022, 23:13:41) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from boletin.models import Registrado
>>> gente = Registrado.objects.all()
>>> gente
<QuerySet []>
>>> persona1 = Registrado.objects.create(nombre='Fran',email='franciscojavier.delvalle.rodriguez.es')
  File "<console>", line 1
    persona1 = Registrado.objects.create(nombre='Fran',email='franciscojavier.delvalle.rodriguez.es')^
SyntaxError: unmatched ')'
>>> persona1 = Registrado.objects.create(nombre='Fran',email='franciscojavier.delvalle.rodriguez.es')
>>> persona1
<Registrado: franciscojavier.delvalle.rodriguez.es>
>>> -
```

En el archivo admin.py vamos a realizar una importación con “from .models import Registrado” y añadimos la linea “admin.site.register(Registrado)”



```
settings.py      models.py      admin.py
  x   x   x
1  from django.contrib import admin
2
3  # Register your models here.
4  from .models import Registrado
5
6  admin.site.register(Registrado)
```

Una vez hecho recargamos la página y podemos ver que se ha creado correctamente

Administración de Django

BIENVENIDOS, FRAN. VER EL SITIO / CAMBIAR CONTRASEÑA / CERRAR SESIÓN

Sitio administrativo

AUTENTICACIÓN Y AUTORIZACIÓN

Grupos	+ Añadir	Modificar
Usuarios	+ Añadir	Modificar

BOLETIN

Registrados	+ Añadir	Modificar
-----------------------------	--------------------------	---------------------------

Acciones recientes

Mis acciones

[+ usuario](#)
Usuario

Inicio > Boletín > Registrados

Empiece a escribir para filtrar...

AUTENTICACIÓN Y AUTORIZACIÓN

Grupos	+ Añadir
Usuarios	+ Añadir

BOLETIN

Registrados	+ Añadir
-----------------------------	--------------------------

«

Seleccione registrado a modificar

Acción: Ir seleccionados 0 de 1

REGISTRADO

franciscojavier.delvalle.rodriguez.es

1 registrado

AÑADIR REGISTRAD

Si queremos que nos salga el nombre en lugar del email modificaremos el fichero models.py y recargamos la página.

The screenshot shows a code editor with three tabs: settings.py, models.py, and admin.py. The models.py tab is active, displaying the following Python code:

```
1 from __future__ import unicode_literals
2
3 from django.db import models
4
5 # Create your models here.
6
7 class Registrado(models.Model):
8     nombre = models.CharField(max_length=100, blank=True, null=True)
9     email = models.EmailField()
10    timestamp = models.DateTimeField(auto_now_add=True, auto_now=False)
11
12    def __unicode__(self):
13        return self.nombre
14
15    def __str__(self):
16        return self.nombre
```

Below the code editor is a screenshot of the Django admin interface. The left sidebar shows a navigation tree with 'Inicio', 'Boletín', and 'Registros'. Under 'BOLETIN', 'Registros' is selected and highlighted in yellow. The main content area is titled 'Seleccione registrado a modificar' and contains a table with one row:

<input type="checkbox"/> REGISTRADO	Fran
1 registrado	

Buttons for 'AÑADIR REGISTRO' and 'Ir' are visible at the top right of the table.

En mi caso lo vamos a dejar con el email.

8. Personalizar modelo en el admin

Personalizamos el display de la Administración de Django, para ello vamos al fichero 'admin.py' y debajo de la importación vamos a escribir un class que será 'AdminRegistrado', con varias listas:

Además, dentro de 'AdminRegistrado' creamos otra clase 'Meta' con 'model = Registrado'

```
1 from django.contrib import admin
2
3 # Register your models here.
4 from .models import Registrado
5
6 class AdminRegistrado(admin.ModelAdmin):
7     list_display = ["email", "nombre", "timestamp"]
8     #list_display_links = ["nombre"]
9     list_filter = ["timestamp"]
10    list_editable = ["nombre"]
11    search_fields = ["email", "nombre"]
12    class Meta:
13        model = Registrado
14
15 admin.site.register(Registrado, AdminRegistrado)
```

El resultado sería el siguiente

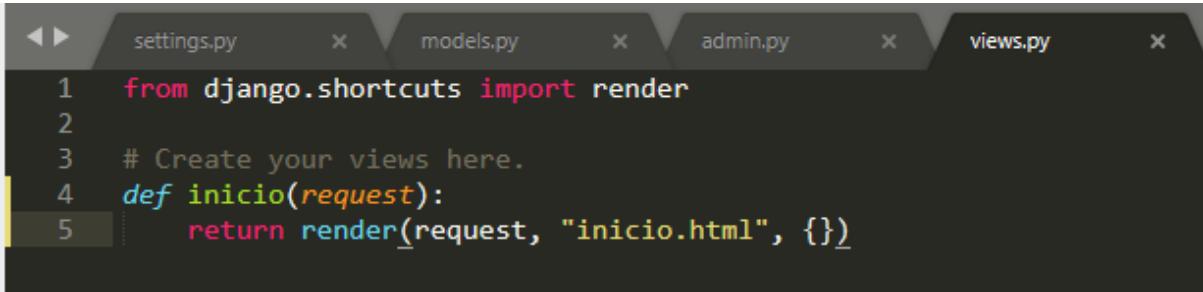
Seleccione registrado a modificar

<input type="checkbox"/>	EMAIL	NOMBRE	TIMESTAMP
<input type="checkbox"/>	franciscojavier.delvalle.rodriguez.es	Fran	20 de Abril de 20

1 registrado Guardar

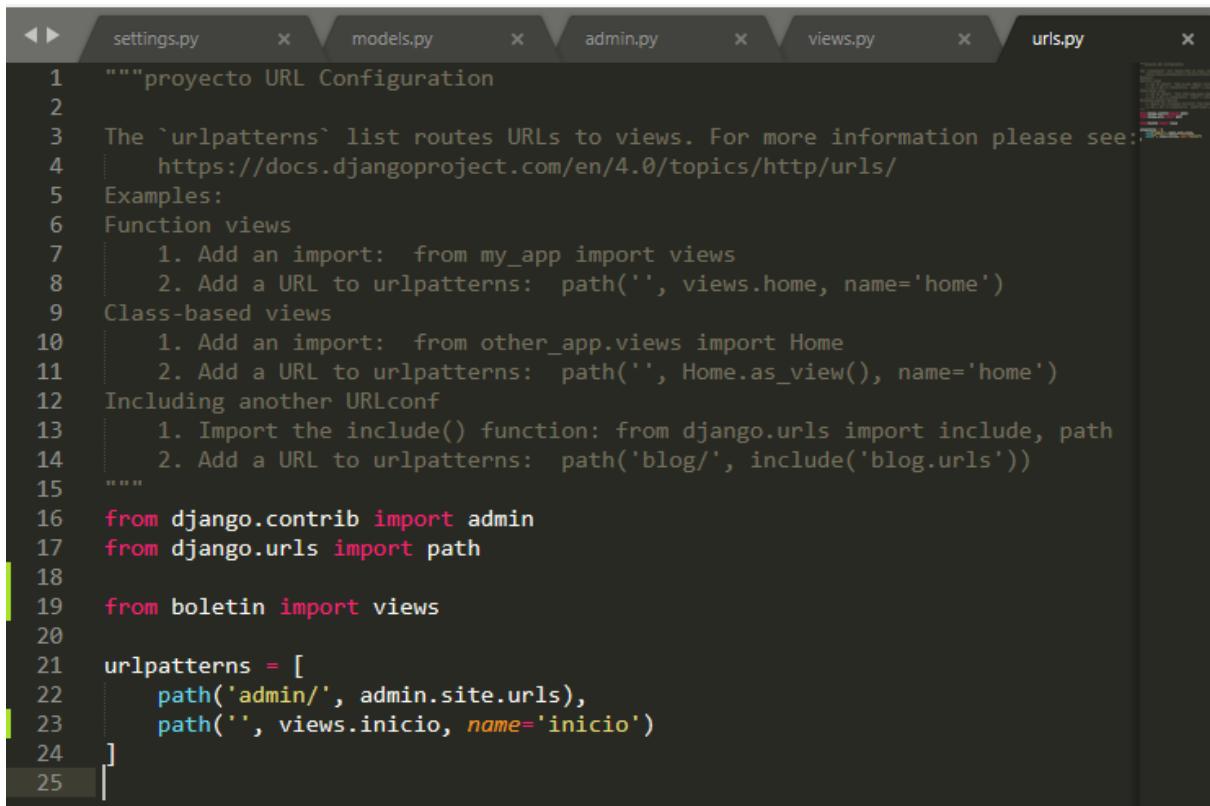
9. Primera vista

Escribimos nuestra primera vista, para ello nos vamos al fichero “views.py” y escribimos lo siguiente:



```
from django.shortcuts import render
# Create your views here.
def inicio(request):
    return render(request, "inicio.html", {})
```

Como podemos observar, hemos añadido una url como plantilla para la vista, pero no está configurada, y vamos a abrir el fichero llamado “urls.py”



```
""" proyecto URL Configuration
The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/4.0/topics/http/urls/
Examples:
Function views
    1. Add an import: from my_app import views
    2. Add a URL to urlpatterns: path('', views.home, name='home')
Class-based views
    1. Add an import: from other_app.views import Home
    2. Add a URL to urlpatterns: path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path
from boletin import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.inicio, name='inicio')
]
```

Importamos nuestra vista ‘from boletin import views’ y abajo añadimos la ruta de nuestra vista, de nombre ‘inicio’. Si lo dejamos así e intentamos entrar en la página, vemos que no podemos acceder debido a que no tenemos una plantilla “inicio.html” creada.

TemplateDoesNotExist at /

inicio.html

```
Request Method: GET
Request URL: http://127.0.0.1:8000/
Django Version: 4.0.4
Exception Type: TemplateDoesNotExist
Exception Value: inicio.html
Exception Location: C:\Users\usuario\Desktop\proyecto\lib\site-packages\django\template\loader.py, line 19, in get_template
Python Executable: C:\Users\usuario\Desktop\proyecto\Scripts\python.exe
Python Version: 3.10.4
Python Path: ['c:\\\\users\\\\usuario\\\\Desktop\\\\proyecto\\\\src', 'c:\\\\users\\\\usuario\\\\AppData\\\\Local\\\\Programs\\\\Python\\\\Python310\\\\python310.zip', 'c:\\\\users\\\\usuario\\\\AppData\\\\Local\\\\Programs\\\\Python\\\\Python310\\\\DLLs', 'c:\\\\users\\\\usuario\\\\AppData\\\\Local\\\\Programs\\\\Python\\\\Python310\\\\lib', 'c:\\\\users\\\\usuario\\\\AppData\\\\Local\\\\Programs\\\\Python\\\\Python310', 'c:\\\\users\\\\usuario\\\\Desktop\\\\proyecto', 'c:\\\\users\\\\usuario\\\\Desktop\\\\proyecto\\\\lib\\\\site-packages']
Server time: Sat, 23 Apr 2022 10:15:25 +0000
```

Template-loader postmortem

Django tried loading these templates, in this order:

Using engine django:

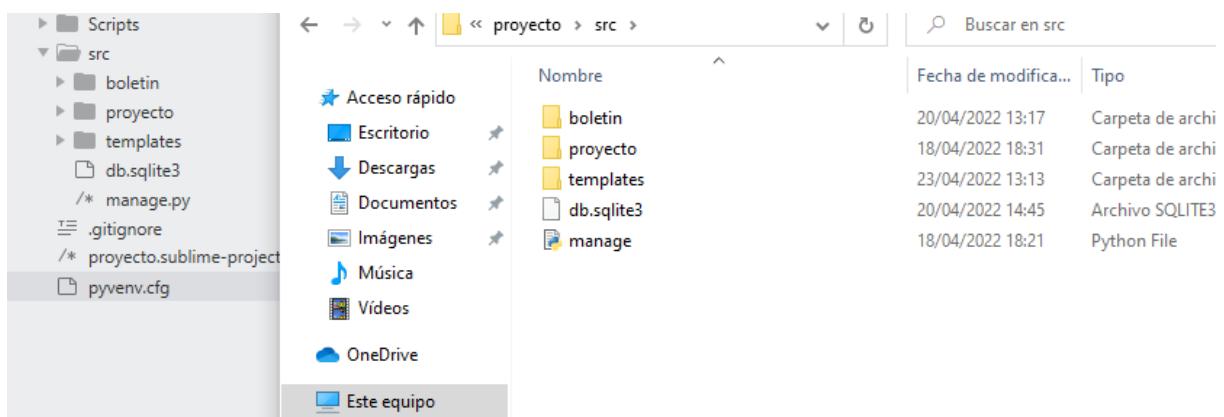
- django.template.loaders.app_directories.Loader: C:\Users\usuario\Desktop\proyecto\lib\site-packages\django\contrib\admin\templates\inicio.html (Source does not exist)
- django.template.loaders.app_directories.Loader: C:\Users\usuario\Desktop\proyecto\lib\site-packages\django\contrib\auth\templates\inicio.html (Source does not exist)

10. Configuración de plantillas

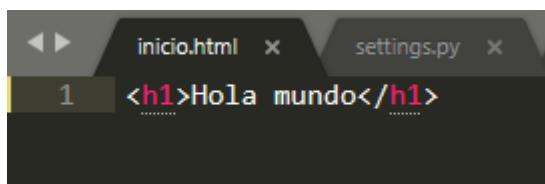
Las plantillas están guardadas en el fichero "setting.py" y añadiremos la linea " 'DIRS': [BASE_DIR / 'templates']"

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [BASE_DIR / 'templates'],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages'
            ],
        },
    },
]
```

Creamos un directorio nuevo dentro del directorio src, el cual se llamará templates, donde crearemos el html.



Creamos en el directorio templates un fichero "inicio.html" en el cual escribimos algo, en mi caso "Hola mundo"



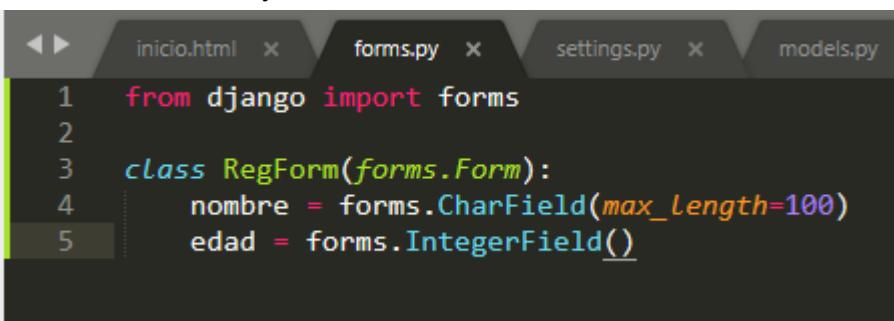
Finalmente recargamos la página y vemos lo siguiente



Hola mundo

11. Escribir Formulario.

Dentro de nuestra aplicación creamos un nuevo fichero que lo llamaremos “forms.py” Tenemos que hacer una importación y escribimos nuestro formulario, en este caso podemos añadir campos sin necesidad de que este exista en nuestro modelo. En este caso solo ponemos el nombre y la edad.



```
1 from django import forms
2
3 class RegForm(forms.Form):
4     nombre = forms.CharField(max_length=100)
5     edad = forms.IntegerField()
```

12. Formulario en una vista

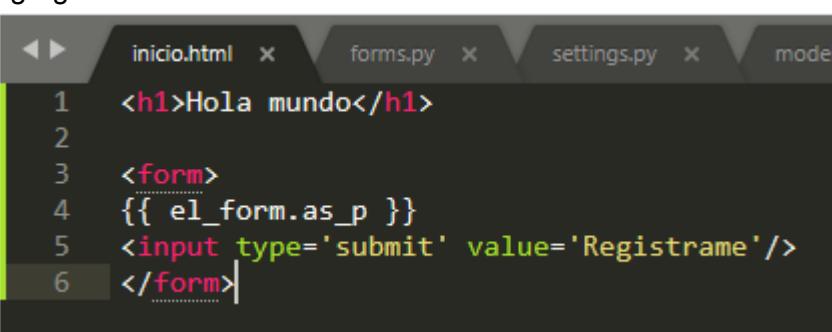
Si queremos añadir el formulario a nuestra vista abrimos el fichero 'views.py' e importamos RegForm

```
from django.shortcuts import render

from .forms import RegForm

# Create your views here.
def inicio(request):
    form = RegForm()
    context = {
        "el_form": form,
    }
    return render(request, "inicio.html", context)
```

Creamos una variable nueva “form = RegForm()” y un diccionario context, que lo agregaremos a nuestro html.



```
1 <h1>Hola mundo</h1>
2
3 <form>
4 {{ el_form.as_p }}
5 <input type='submit' value='Registrate' />
6 </form>
```

Dentro de inicio.html cargaremos la variable y lo ponemos para que nos lo muestre en formato párrafo, finalmente le crearemos un botón el cual lo llamaremos Registrarme. Finalmente cargaremos la página y veremos el siguiente resultado.



Hola mundo

Nombre:

Edad:

Regístrate

13. Método HTTP POST en formulario

Antes de empezar tenemos que saber que CRUD (Create Read Update delete) consiste en 4 funciones principales de cada aplicación web y cada uno tiene método HTTP para llevar a cabo la función.

Lo que se pretende es guardar los datos que se introduzcan en el formulario creado anteriormente en la base de datos, para ello utilizaremos métodos.

Si nosotros introducimos datos en el formulario y pinchamos sobre el botón, observamos que no ocurre nada, vemos que aparece en el navegador como método GET pero no se almacena en ningún objeto.

← → ⌛ ⓘ 127.0.0.1:8000/?nombre=Fran&edad=22

Hola mundo

Nombre:

Edad:

Regístrate

Para arreglar esto, volvemos a nuestro 'inicio.html' y dentro de la etiqueta 'form' añadimos el método POST. Si queremos utilizar POST tenemos que añadir el csrf_token a nuestro formulario (ayuda contra la falsificación de peticiones).

```
<h1>Hola mundo</h1>
<form method="POST" action="">{% csrf_token %}
{{ el_form.as_p }}
<input type='submit' value='Registrate' />
</form>
```

Si volvemos a añadir nuestro nombre y edad, observamos en el cmd que aparece el método POST.

```
[23/Apr/2022 14:14:17] "GET /?nombre=Fran&edad=22 HTTP/1.1" 200 357
[23/Apr/2022 14:20:08] "GET / HTTP/1.1" 200 503
[23/Apr/2022 14:20:15] "POST / HTTP/1.1" 200 503
```

14. Validaciones formulario pt.1

Para analizar si los datos del formulario son datos limpios, primero tendremos que hacer algo con nuestro método POST en 'views.py' Vamos a añadir a nuestro formulario la petición del POST (request.POST)

```
from django.shortcuts import render

from .forms import RegForm

# Create your views here.
def inicio(request):
    form = RegForm(request.POST)
    context = {
        "el_form": form,
    }
    return render(request, "inicio.html", context)
```

Hola mundo

- Este campo es obligatorio.

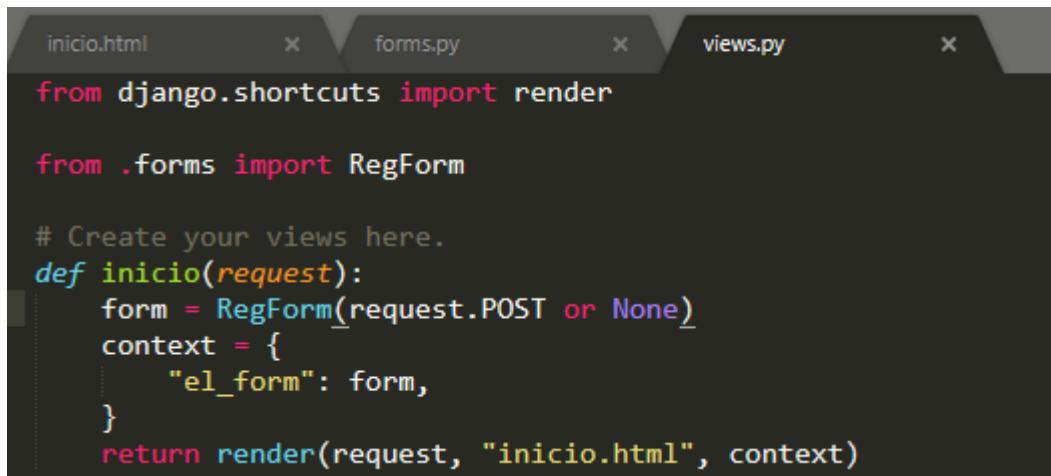
Nombre:

- Este campo es obligatorio.

Edad:

[Regístrate](#)

Si recargamos la web veremos que nos indica los dos campos como obligatorios para llenar. En el modelo el campo de 'nombre' no es obligatorio pero en el formulario si. Si queremos quitar ese mensaje, basta con incluir 'or None' dentro de 'form'.



```
inicio.html      forms.py      views.py
x               x               x

from django.shortcuts import render

from .forms import RegForm

# Create your views here.
def inicio(request):
    form = RegForm(request.POST or None)
    context = {
        "el_form": form,
    }
    return render(request, "inicio.html", context)
```

Podemos hacer que el formulario saque los datos limpios cuando nos registremos, para ello ponemos la condición en 'views.py' de que si el formulario es válido, nos muestre nuestros datos de manera clara.

```
[23/Apr/2022 15:18:10] "GET / HTTP/1.1" 200 503
{'nombre': 'Fran', 'edad': 22}
[23/Apr/2022 15:18:16] "POST / HTTP/1.1" 200 527
```

Si queremos quitar el diccionario, volvemos a escribir:

```
# Create your views here.
def inicio(request):
    form = RegForm(request.POST or None)
    if form.is_valid():
        form_data = form.cleaned_data
        print (form_data.get("nombre"))
        print (form_data.get("edad"))
    context = {
        "el_form": form,
    }
    return render(request, "inicio.html", context)
```

```
[23/Apr/2022 15:22:23] "GET / HTTP/1.1" 200 503
Fran
22
[23/Apr/2022 15:22:32] "POST / HTTP/1.1" 200 527
```

15. Guardar datos del formulario con el modelo.

Ahora vamos a guardar objetos con los datos del formulario utilizando nuestro modelo. Lo primero que vamos a hacer es ir al fichero “forms.py” y sustituir el campo “edad” por “email”

```
from django.shortcuts import render

from .forms import RegForm
from .models import Registrado

# Create your views here.
def inicio(request):
    form = RegForm(request.POST or None)
    if form.is_valid():
        form_data = form.cleaned_data
        abc = form_data.get("email")
        obj = Registrado.objects.create(email=abc)
    context = {
        "el_form": form,
    }
    return render(request, "inicio.html", context)
```

Recargamos la página y pondremos un nombre y un correo, veremos que en registrado se guarda el correo pero no el nombre, ya que en la vista no hemos hecho que aparezca.

Seleccione registrado a modificar

EMAIL	NOMBRE	TIMESTAMP
hola@hola.com		23 de Abr
franciscojavier.delvalle.rodriguez.es	Fran	20 de Abr

Acción: ----- Ir seleccionados 0 de 2

2 registrados Guardar

Para que podamos ver el nombre volvemos a modificar el fichero “views.py” quedando de la siguiente manera.

```
inicio.html      forms.py      views.py
from django.shortcuts import render

from .forms import RegForm
from .models import Registrado

# Create your views here.
def inicio(request):
    form = RegForm(request.POST or None)
    if form.is_valid():
        form_data = form.cleaned_data
        abc = form_data.get("email")
        abc2 = form_data.get("nombre")
        obj = Registrado.objects.create(email=abc, nombre=abc2)
    context = {
        "el_form": form,
    }
    return render(request, "inicio.html", context)
```

A continuación vamos a la página y añadimos otro nombre y email, podemos observar que ahora se guarda también el nombre.

Inicio > Boletín > Registrados

Seleccione registrado a modificar

<input type="checkbox"/>	EMAIL	NOMBRE	TIMESTAM
<input type="checkbox"/>	persona@me.com	persona	23 de Abr
<input type="checkbox"/>	hola@hola.com		23 de Abr
<input type="checkbox"/>	franciscojavier.delvalle.rodriguez.es	Fran	20 de Abr

3 registrados Guardar

16. Model Form

Modificamos el fichero “forms.py” para crear nuestro propio model form, en el cual solo tendrá el campo email.

```

inicio.html      x   forms.py      x   admin.py
from django import forms

from .models import Registrado

class RegModelForm(forms.ModelForm):
    class Meta:
        model = Registrado
        fields = ["email"]

class RegForm(forms.Form):
    nombre = forms.CharField(max_length=100)
    email = forms.EmailField()

```

También lo tendremos que poner en el fichero “admin.py” quedando de la siguiente forma.

```
inicio.html      x     forms.py      x     admin.py      x
from django.contrib import admin

# Register your models here.
from .forms import RegModelForm
from .models import Registrado

class AdminRegistrado(admin.ModelAdmin):
    list_display = ["email", "nombre", "timestamp"]
    form = RegModelForm
    #list_display_links = ["nombre"]
    list_filter = ["timestamp"]
    list_editable = ["nombre"]
    search_fields = ["email", "nombre"]
    #class Meta:
    #    model = Registrado

admin.site.register(Registrado, AdminRegistrado)
```

En la página quedaría de la siguiente manera:

Inicio > Boletín > Registrados > Añadir registrado

Añadir registrado

Email:

Para que aparezca el nombre volvemos a modificar el fichero “forms.py“.

```
inicio.html      x     forms.py      x     admin.py      x
from django import forms

from .models import Registrado

class RegModelForm(forms.ModelForm):
    class Meta:
        model = Registrado
        fields = ["nombre", "email"]

    class RegForm(forms.Form):
        nombre = forms.CharField(max_length=100)
        email = forms.EmailField()
```

Esto nos da como resultado lo siguiente:

Añadir registrado

Nombre:

Email:

[Guardar y añadir otro](#) [Guardar y continuar editando](#) **GUARDAR**

17. Validaciones Model Form

Como hemos visto anteriormente Django aplica sus propias validaciones según cada campo, si no escribimos un correo en el campo email, no lo dará como válido, teniendo que comprobar si esos datos no son maliciosos para la base de datos.

Para ello, en el fichero “forms.py” crearemos la validaciones, en este caso crearemos un campo de validación llamado “clean_email”

```
inicio.html      x      forms.py      x      admin.py
from django import forms

from .models import Registrado

class RegModelForm(forms.ModelForm):
    class Meta:
        model = Registrado
        fields = ["nombre", "email"]

    def clean_email(self):
        print (self.cleaned_data)
        return "email@email.com"

class RegForm(forms.Form):
    nombre = forms.CharField(max_length=100)
    email = forms.EmailField()
```

A continuación vemos que podemos agregar el correo correctamente

✓ El registrado "email@email.com" fue agregado correctamente.

Seleccione registrado a modificar

AÑADIR R

Buscar

Acción: seleccionados 0 de 4

<input type="checkbox"/> EMAIL	NOMBRE	TIMESTAMP
<input type="checkbox"/> email@email.com		23 de Abr

FILTRO

Por timestamp

Cualquier fecha

Hoy

Últimos 7 días

Este mes

Este año

Vamos a escribirlo de manera que cuando escribamos un correo no educativo (.EDU) nos salga error.

```
inicio.html      x  forms.py      x  admin.py      x  views.py      x
from django import forms

from .models import Registrado

class RegModelForm(forms.ModelForm):
    class Meta:
        model = Registrado
        fields = ["nombre", "email"]

    def clean_email(self):
        email = self.cleaned_data
        if not ".edu" in email:
            raise forms.ValidationError("Por favor utiliza un email con la extensión .EDU")
        return email

class RegForm(forms.Form):
    nombre = forms.CharField(max_length=100)
    email = forms.EmailField()
```

Añadir registrado

Por favor corrija el siguiente error.

Nombre:

Por favor utiliza un email con la extensión .EDU

Email:

fran@me.com

Guardar y añadir otro

Guardar y continuar editando

GUARDAR

Pero esto no impide que pongamos un correo de nombre edu sin la extensión propia, si lo que queremos es que solo acepte la extensión, vamos a crear una nueva variable:

```
inicio.html      forms.py      admin.py      views.py
from django import forms

from .models import Registrado

class RegModelForm(forms.ModelForm):
    class Meta:
        model = Registrado
        fields = ["nombre", "email"]

    def clean_email(self):
        email = self.cleaned_data.get("email")
        email_base, proveedor = email.split("@")
        dominio, extension = proveedor.split(".")
        if not extension == ".edu":
            raise forms.ValidationError("Por favor utiliza un email con la extension .EDU")
        return email

class RegForm(forms.Form):
    nombre = forms.CharField(max_length=100)
    email = forms.EmailField()
```

Añadir registrado

Por favor corrija el siguiente error.

Nombre:

Por favor utiliza un email con la extension .EDU

Email:

Guardar y añadir otro

Guardar y continuar editando

GUARDAR

Si queremos crear una validación para el nombre modificamos el fichero “forms.py” de la siguiente manera.

```
inicio.html      forms.py      admin.py      views.py
from django import forms

from .models import Registrado

class RegModelForm(forms.ModelForm):
    class Meta:
        model = Registrado
        fields = ["nombre", "email"]

    def clean_email(self):
        email = self.cleaned_data.get("email")
        email_base, proveedor = email.split("@")
        dominio, extension = proveedor.split(".")
        if not extension == ".edu":
            raise forms.ValidationError("Por favor utiliza un email con la extension .EDU")
        return email

    def clean_nombre(self):
        nombre = self.cleaned_data.get("nombre")
        #validaciones
        return nombre

class RegForm(forms.Form):
    nombre = forms.CharField(max_length=100)
    email = forms.EmailField()
```

18. Contexto en la vista, plantillas

A continuación veremos en detalle cómo funciona el contexto, las variables tanto en las plantillas como en las vistas.

En nuestra vista, vamos a agregar un título para nuestro html.

```
inicio.html      views.py
from django.shortcuts import render

from .forms import RegForm
from .models import Registrado

# Create your views here.
def inicio(request):
    titulo = "HOLA"
    form = RegForm(request.POST or None)
    if form.is_valid():
        form_data = form.cleaned_data
        abc = form_data.get("email")
        abc2 = form_data.get("nombre")
        obj = Registrado.objects.create(email=abc, nombre=abc2)
    context = {
        "titulo": titulo,
        "el_form": form,
    }
    return render(request, "inicio.html", context)
```

A parte, lo añadiremos a 'inicio.html' haciendo referencia a 'titulo'.

```
inicio.html           x     views.py           x
<!--<h1>Hola mundo</h1> -->
{{ titulo }}
<hr/>
<br/>

<form method="POST" action="">{{ csrf_token %}}
{{ el_form.as_p }}
<input type='submit' value='Registrate' />
</form>
```

HOLA

Nombre:

Email:

También podemos hacer que si el usuario no esta autenticado con su cuenta, le aparezca otro mensaje de bienvenida distinto:

```
inicio.html           x     views.py           x
from django.shortcuts import render

from .forms import RegForm
from .models import Registrado

# Create your views here.
def inicio(request):
    titulo = "HOLA"
    if request.user.is_authenticated:
        titulo = "Bienvenido %s" %(request.user)
    form = RegForm(request.POST or None)
    if form.is_valid():
        form_data = form.cleaned_data
        abc = form_data.get("email")
        abc2 = form_data.get("nombre")
        obj = Registrado.objects.create(email=abc, nombre=abc2)
    context = {
        "titulo": titulo,
        "el_form": form,
    }
    return render(request, "inicio.html", context)
```

Bienvenido Fran

Nombre:

Email:

Para jugar un poco más con nuestra plantilla, podemos agregar diferentes variables.



The screenshot shows a code editor with two tabs open: 'inicio.html' and 'views.py'. The 'inicio.html' tab contains the following HTML code:

```
<!--<h1>Hola mundo</h1> -->
{{ titulo }}
<br/>
{{ request.user }}
<br/>
variable = {{ abc }}
<hr/>
<br/>

<form method="POST" action="">{% csrf_token %}
{{ el_form.as_p }}
<input type='submit' value='Regístrate' />
</form> |
```

Bienvenido Fran

Fran

variable =

Nombre:

Email:

Donde "request.user" somos nosotros y "abc" una variable que no está definida.

19. Model Form en la vista

En nuestra vista renderizamos RegForm desde el fichero "forms.py", se trata de un Custom Form. A continuación vamos a sustituirlo por nuestro Model Form, esto lo hacemos para guardar objetos nuevos usando el modelo.

Importamos 'RegModelForm' en la vista y creamos una instancia.

The screenshot shows a terminal window with two tabs open: 'views.py' and 'forms.py'. The 'views.py' tab contains Python code for a view named 'inicio'. The code imports 'render' from 'django.shortcuts', 'RegForm' and 'RegModelForm' from 'forms', and 'Registrado' from 'models'. It defines a function 'inicio' that sets the title to 'HOLA'. If the user is authenticated, it sets the title to 'Bienvenido %s' where %s is the user's name. It creates a 'RegModelForm' instance from the request's POST data or None. If the form is valid, it saves the instance with commit=False, prints the instance and its timestamp, and creates a new 'Registrado' object with email and nombre fields. It then creates a context dictionary with 'titulo' and 'el_form' keys, and returns the rendered 'inicio.html' template. The 'forms.py' tab shows the definition of the 'RegModelForm' class which inherits from 'ModelForm' and specifies the 'Registrado' model and field names. Below the terminal window, there is a command-line log showing a GET request at 18:00:07 and a POST request at 18:00:20. The POST request shows the user 'hola@hola.edu' and a timestamp 'None'.

```
from django.shortcuts import render

from .forms import RegForm, RegModelForm
from .models import Registrado

# Create your views here.
def inicio(request):
    titulo = "HOLA"
    if request.user.is_authenticated:
        titulo = "Bienvenido %s" %(request.user)
    form = RegModelForm(request.POST or None)
    if form.is_valid():
        instance = form.save(commit=False)
        print (instance)
        print (instance.timestamp)
        #form_data = form.cleaned_data
        #abc = form_data.get("email")
        #abc2 = form_data.get("nombre")
        #obj = Registrado.objects.create(email=abc, nombre=abc2)
    context = {
        "titulo": titulo,
        "el_form": form,
    }
    return render(request, "inicio.html", context)

quit the server with CTRL-BREAK.
[23/Apr/2022 18:00:07] "GET / HTTP/1.1" 200 560
hola@hola.edu
None
[23/Apr/2022 18:00:20] "POST / HTTP/1.1" 200 595
```

Podemos observar en la línea de comando que nos muestra el correo (recordad que nuestro unicode era 'email') pero no muestra el timestamp. Eso es debido a que tenemos qué 'commit=False', eso impide que ese registrado nuevo se guarde, nos falta una línea más, 'instance.save()'

```
inicio.html      x      views.py      x      forms.py      x
from django.shortcuts import render

from .forms import RegForm, RegModelForm
from .models import Registrado

# Create your views here.
def inicio(request):
    titulo = "HOLA"
    if request.user.is_authenticated:
        titulo = "Bienvenido %s" %(request.user)
    form = RegModelForm(request.POST or None)
    if form.is_valid():
        instance = form.save(commit=False)
        instance.save()
        print (instance)
        print (instance.timestamp)
        #form_data = form.cleaned_data
        #abc = form_data.get("email")
        #abc2 = form_data.get("nombre")
        #obj = Registrado.objects.create(email=abc, nombre=abc2)
    context = {
        "titulo": titulo,
        "el_form": form,
    }
    return render(request, "inicio.html", context)
```

```
hola@hola.edu
2022-04-23 16:03:30.451659+00:00
[23/Apr/2022 18:03:30] "POST / HTTP/1.1" 200 595
```

Confirmamos que el timestamp se acaba de guardar. Si os acordáis, en 'models.py' el campo 'nombre' no era obligatorio. Vamos a añadir una linea mas en nuestro modelo, para que en caso de que no se ponga un nombre de usuario, se agregue otro por defecto.

```
inicio.html      x  views.py      x  forms.py      x  models.py
from django.shortcuts import render

from .forms import RegForm, RegModelForm
from .models import Registrado

# Create your views here.
def inicio(request):
    titulo = "HOLA"
    if request.user.is_authenticated:
        titulo = "Bienvenido %s" %(request.user)
    form = RegModelForm(request.POST or None)
    if form.is_valid():
        instance = form.save(commit=False)
        if not instance.nombre:
            instance.nombre = "PERSONA"
        instance.save()
        print (instance)
        print (instance.timestamp)
        #form_data = form.cleaned_data
        #abc = form_data.get("email")
        #abc2 = form_data.get("nombre")
        #obj = Registrado.objects.create(email=abc, nombre=abc2)
    context = {
        "titulo": titulo,
        "el_form": form,
    }
    return render(request, "inicio.html", context)
```

Inicio > Boletín > Registrados

Seleccione registrado a modificar

<input type="checkbox"/>	EMAIL	NOMBRE	TIMESTAM
<input type="checkbox"/>	hola@hola.edu	PERSONA	23 de /

Ponemos el “commit=False” para que en caso de que no añadieran el nombre este automáticamente pondrá PERSONA para guardarlo.

Otra cosa que podemos hacer es modificar el contexto. Vamos a hacer que cuando nos registremos aparezca un mensaje con el nombre que se haya introducido. En caso de que no haya un nombre, podemos hacer que nos muestre el correo registrado.

```
inicio.html      x  views.py      x  forms.py      x  models.py

from django.shortcuts import render

from .forms import RegForm, RegModelForm
from .models import Registrado

# Create your views here.
def inicio(request):
    titulo = "HOLA"
    if request.user.is_authenticated:
        titulo = "Bienvenido %s" %(request.user)
    form = RegModelForm(request.POST or None)

    context = {
        "titulo": titulo,
        "el_form": form,
    }

    if form.is_valid():
        instance = form.save(commit=False)
        nombre = form.cleaned_data.get("nombre")
        email = form.cleaned_data.get("email")
        if not instance.nombre:
            instance.nombre = "PERSONA"
        instance.save()

        context = {
            "titulo" : "Gracias %s!" %(nombre)
        }
        print (instance)
        print (instance.timestamp)
        #form_data = form.cleaned_data
        #abc = form_data.get("email")
        #abc2 = form_data.get("nombre")
        #obj = Registrado.objects.create(email=abc, nombre=abc2)

    return render(request, "inicio.html", context)
```

Gracias Fran!
Fran

[Regístrate](#)

Si la persona añade un correo pero no un nombre podemos hacer lo siguiente:

```
inicio.html      x  views.py      x  forms.py      x  models.py
from django.shortcuts import render

from .forms import RegForm, RegModelForm
from .models import Registrado

# Create your views here.
def inicio(request):
    titulo = "HOLA"
    if request.user.is_authenticated:
        titulo = "Bienvenido %s" %(request.user)
    form = RegModelForm(request.POST or None)

    context = {
        "titulo": titulo,
        "el_form": form,
    }

    if form.is_valid():
        instance = form.save(commit=False)
        nombre = form.cleaned_data.get("nombre")
        email = form.cleaned_data.get("email")
        if not instance.nombre:
            instance.nombre = "PERSONA"
        instance.save()

        context = {
            "titulo" : "Gracias %s!" %(nombre)
        }
        if not nombre:
            context = {
                "titulo" : "Gracias %s!" %(email)
            }
    print (instance)
    print (instance.timestamp)
    #form_data = form.cleaned_data
    #abc = form_data.get("email")
    #abc2 = form_data.get("nombre")
    #obj = Registrado.objects.create(email=abc, nombre=abc2)

    return render(request, "inicio.html", context)
```

Activar Windows

Vea la Configuración para más información.

Gracias hola@hola.edu!
Fran

[Regístrate](#)

Vemos que aún después de habernos registrado aparece ese botón de 'Regístrate'. Vamos a hacer que no se vea. Habrá que editar nuestra plantilla.

The screenshot shows a code editor with three tabs: 'inicio.html', 'views.py', and 'forms.py'. The 'inicio.html' tab displays the following Django template code:

```
<!--<h1>Hola mundo</h1> -->
{{ titulo }}
<br/>
{{ request.user }}
<hr/>
<br/>

{% if form %}
<form method="POST" action="">{{ csrf_token }}
{{ el_form.as_p }}
<input type='submit' value='Regístrate' />
</form>
{% endif %}
```

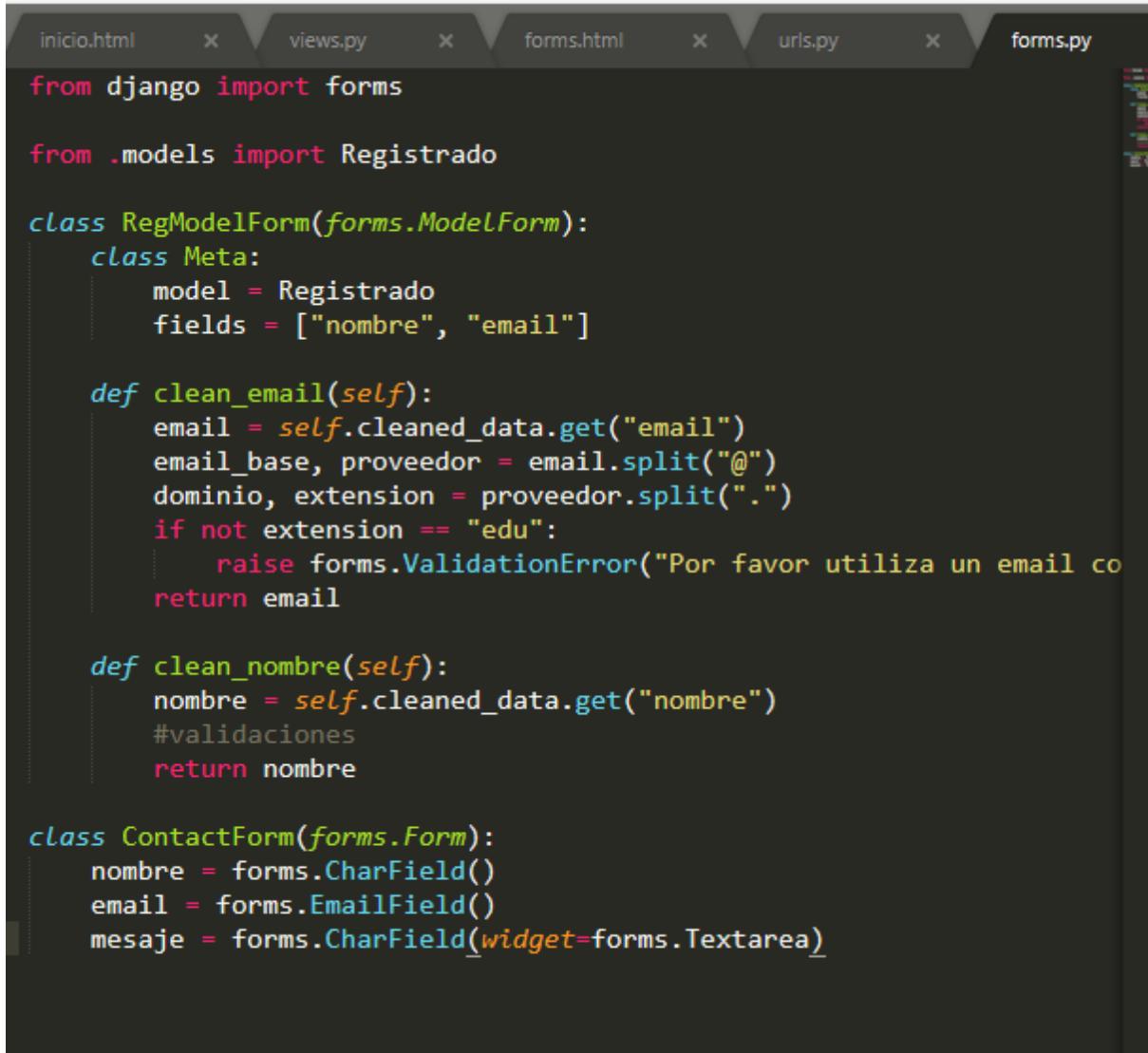
The 'views.py' tab contains a simple view function:

```
def inicio(request):
    return render(request, 'inicio.html')
```

The browser output on the right shows the rendered page with the text 'Gracias hola@hola.edu!' and 'Fran' below it, indicating that the registration button was present in the original template but has been removed.

20. Custom Form para contacto

Para nuestra página principal de registro usamos el Model Form, pero en 'forms.py' teníamos un 'RegForm' que lo vamos a convertir en un formulario de contacto, por si alguien se quiere poner en contacto con nosotros.



The screenshot shows a code editor with several tabs at the top: 'inicio.html', 'views.py', 'forms.html', 'urls.py', and 'forms.py'. The 'forms.py' tab is active, displaying the following Python code:

```
from django import forms
from .models import Registrado

class RegModelForm(forms.ModelForm):
    class Meta:
        model = Registrado
        fields = ["nombre", "email"]

    def clean_email(self):
        email = self.cleaned_data.get("email")
        email_base, proveedor = email.split("@")
        dominio, extension = proveedor.split(".")
        if not extension == "edu":
            raise forms.ValidationError("Por favor utiliza un email con .edu")
        return email

    def clean_nombre(self):
        nombre = self.cleaned_data.get("nombre")
        #validaciones
        return nombre

class ContactForm(forms.Form):
    nombre = forms.CharField()
    email = forms.EmailField()
    mensaje = forms.CharField(widget=forms.Textarea)
```

El widget ese nos vale para que el mensaje sea mas grande, tenga mas espacio para escribirse.

Aparte, tendremos que crear una vista nueva para dicho contacto (recordad que hay que borrar la importación antigua y crear una nueva para ContactForm)

```
inicio.html      x     views.py      x     forms.html      x     urls.py      x     forms.py
instance = form.save(commit=False)
nombre = form.cleaned_data.get("nombre")
email = form.cleaned_data.get("email")
if not instance.nombre:
    instance.nombre = "PERSONA"
instance.save()

context = {
    "titulo" : "Gracias %s!" %(nombre)
}
if not nombre:
    context = {
        "titulo" : "Gracias %s!" %(email)
    }
print (instance)
print (instance.timestamp)
#form_data = form.cleaned_data
#abc = form_data.get("email")
#abc2 = form_data.get("nombre")
#obj = Registrado.objects.create(email=abc, nombre=abc2)

return render(request, "inicio.html", context)

def contact(request):
    form = ContactForm(request.POST or None)
    context = {
        "form": form,
    }
    return render(request, "forms.html", context)
```

Dentro de las plantillas vamos a crear una nueva, 'forms.html' y agregar dicha a 'urls.py'

```
inicio.html      x     views.py      x     forms.html      x
{{ titulo }}
<br/>
{{ request.user }}
<hr/>
<br/>
|
<form method="POST" action="">{% csrf_token %}
{{ form.as_p }}
<input type='submit' value='Registrate' />
</form>
```

```
inicio.html      x     views.py      x     forms.html      x     urls.py      x
"""
projecto URL Configuration

The `urlpatterns` list routes URLs to views. For more information
    https://docs.djangoproject.com/en/4.0/topics/http/urls/
Examples:
Function views
    1. Add an import: from my_app import views
    2. Add a URL to urlpatterns: path('', views.home, name='
Class-based views
    1. Add an import: from other_app.views import Home
    2. Add a URL to urlpatterns: path('', Home.as_view(), na
Including another URLconf
    1. Import the include() function: from django.urls import
    2. Add a URL to urlpatterns: path('blog/', include('blog
.....
from django.contrib import admin
from django.urls import path

from boletin import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('contact/', views.contact, name='contact'),
    path('', views.inicio, name='inicio'),
]
```

Nos vamos a la página y vemos lo siguiente:

← → ⌂ ⓘ 127.0.0.1:8000/contact/

Fran

Nombre:

Email:

Mensaje:

Agregar unas validaciones

```
def contact(request):
    form = ContactForm(request.POST or None)
    if form.is_valid():
        email = form.cleaned_data.get("email")
        mensaje = form.cleaned_data.get("mensaje")
        nombre = form.cleaned_data.get("nombre")
        print (email, mensaje, nombre)
    context = {
        "form": form,
    }
    return render(request, "forms.html", context)
```

Fran

Nombre:

Email:

Mensaje:

HOLA

pepe@pepe.edu None pepe
[23/Apr/2022 18:54:04] "POST /contact/ HTTP/1.1" 200 680

Si tenemos muchos campos y no queremos escribirlos todos podemos hacer:

```
inicio.html      ✘  views.py      ✘  forms.html      ✘  urls.py
return render(request, "inicio.html", context)

def contact(request):
    form = ContactForm(request.POST or None)
    if form.is_valid():
        for key in form.cleaned_data:
            print (key)
            print (form.cleaned_data.get(key))
            #email = form.cleaned_data.get("email")
            #mensaje = form.cleaned_data.get("mensaje")
            #nombre = form.cleaned_data.get("nombre")
            #print (email, mensaje, nombre)
    context = {
        "form": form,
    }
    return render(request, "forms.html", context)
```

Fran

Nombre:

Email:

HOLA

Mensaje:

```
  Símbolo del sistema - python manage.py runserver
[23/Apr/2022 18:59:56] "POST /contact/ HTTP/1.1" 200 680
nombre
pepe
email
pepe@pepe.edu
mensaje
HOLA
[23/Apr/2022 19:00:52] "POST /contact/ HTTP/1.1" 200 680
-
```

O de la siguiente manera:

```
inicio.html      x      views.py      x      forms.html      x      urls.py
    return render(request, "inicio.html", context)

def contact(request):
    form = ContactForm(request.POST or None)
    if form.is_valid():
        for key, value in form.cleaned_data.items():
            print (key, value)
    #for key in form.cleaned_data:
    #    print (key)
    #    print (form.cleaned_data.get(key))
    #email = form.cleaned_data.get("email")
    #mensaje = form.cleaned_data.get("mensaje")
    #nombre = form.cleaned_data.get("nombre")
    #print (email, mensaje, nombre)
    context = {
        "form": form,
    }
    return render(request, "forms.html", context)
```

Fran

Nombre:

Email:

HOLA

Mensaje:

Regístrate

```
Símbolo del sistema - python manage.py runserver
[23/Apr/2022 19:06:46] "POST /contact/ HTTP/1.1" 200 680
nombre pepe
email pepe@pepe.edu
mensaje HOLA
[23/Apr/2022 19:06:49] "POST /contact/ HTTP/1.1" 200 680
```

21. Configurar email.

Vamos a configurar nuestro correo electrónico para poder enviar mensajes a los usuarios. Vamos a hacer el testing con nuestro formulario de contacto, con dicha información que nos enviaremos nosotros mismos.

Primero de todo hay que agregar opciones en 'settings.py' (en un futuro habrá que desbloquear el captcha de Gmail)

```
EMAIL_HOST = 'smtp.gmail.com'  
EMAIL_HOST_USER = 'tu_email@gmail.com'  
EMAIL_HOST_PASSWORD = 'tupassword'  
EMAIL_PORT = 587  
EMAIL_USE_TLS = True
```

En 'views.py' vamos a ir de nuevo a nuestro 'contact', después de importar nuestros settings (el Fail_silently nos sirve para que muestre el error del servidor ya que estamos haciendo un testing)

```
from django.conf import settings  
from django.core.mail import send_mail
```

Vamos a hacer algunas pruebas {hay que configurar tu cuenta de Gmail para que acepte enviar mensajes desde otras aplicaciones, habilitar que apps tengan acceso a tu cuenta y habilitar el IMAP}

```
settings.py      views.py  
#form_data = form.cleaned_data  
#abc = form_data.get("email")  
#abc2 = form_data.get("nombre")  
#obj = Registrado.objects.create(email=abc, nombre=abc2)  
  
return render(request, "inicio.html", context)  
  
def contact(request):  
    form = ContactForm(request.POST or None)  
    if form.is_valid():  
        #for key, value in form.cleaned_data.items():  
        #    print (key, value)  
        #for key in form.cleaned_data:  
        #    print (key)  
        #    print (form.cleaned_data.get(key))  
        form_email = form.cleaned_data.get("email")  
        form_mensaje = form.cleaned_data.get("mensaje")  
        form_nombre = form.cleaned_data.get("nombre")  
        asunto = 'Form de Contacto'  
        email_from = settings.EMAIL_HOST_USER  
        email_to = [email_from, "otroemail@gmail.com"]  
        email_mensaje = "%s: %s enviado por %s" %(form_nombre, form_mensaje, form_email)  
        send_mail(asunto,  
                  mensaje_email,  
                  email_from,  
                  [email_to],  
                  fail_silently=False  
                )  
        #print (email, mensaje, nombre)  
        context = {  
            "form": form,  
        }  
    return render(request, "forms.html", context)
```

Si dejamos el 'fail_silently' en False, y metemos un correo Gmail falso dará error, pero si lo dejamos en True, intentará conectarse al servidor sin éxito.

22. Configuración de archivos estáticos

Cuando hablamos de archivos estáticos nos referimos a css, imágenes y JavaScrit.

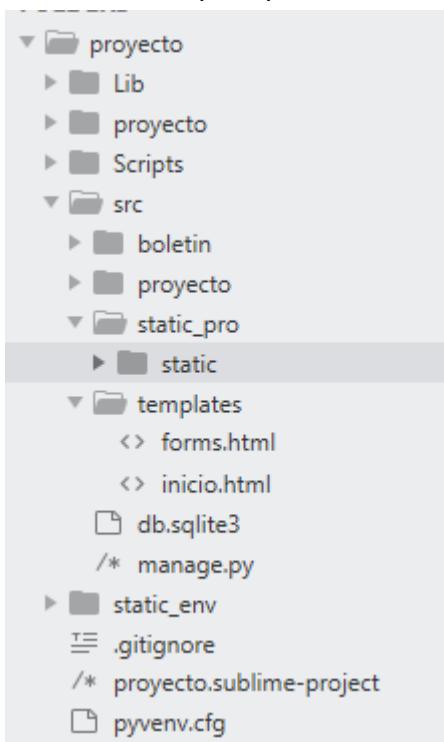
Tenemos que asegurarnos que dentro de 'settings.py' tenemos 'staticfiles' en 'INSTALLED_APPS' y a continuación especificar la ruta donde guardaremos los archivos:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'boletin'
]
```

```
STATIC_URL = 'static/'
#/static/imagenes/img1.jpg

STATICFILE_DIRS = [
    os.path.join(BASE_DIR, "static_pro", "static"),
    '/var/www/static/',
]
```

Vamos a crear una carpeta nueva dentro de nuestra raíz 'src' que se llame 'static_pro', y dentro de ella otra llamada 'static'. Además, dentro de nuestro entorno virtual vamos a añadir una carpeta 'static_env' para trabajar con dichos archivos. Esto nos vale para emular el tener nuestros archivos estáticos en otro servidor para producción.



También tenemos que especificar 'STATIC_ROOT' donde ya vivirán los archivos en producción en otro servidor. Habrá que crear una carpeta 'static_root' dentro de 'static_env'. También podemos hacer lo mismo para 'media', archivos estáticos subidos por terceros.

```
STATIC_URL = 'static/'  
STATIC_URL = 'media/'  
#/static/imagenes/img1.jpg  
  
STATICFILE_DIRS = [  
    os.path.join(BASE_DIR, "static_pro", "static"),  
    #'var/www/static/',  
]  
  
STATIC_ROOT = os.path.join(os.path.dirname(BASE_DIR), "static_env", "static_root")  
MEDIA_ROOT = os.path.join(os.path.dirname(BASE_DIR), "static_env", "media_root")
```

Lo siguiente sería configurar las URLs (solo para desarrollo, para producción no vale)

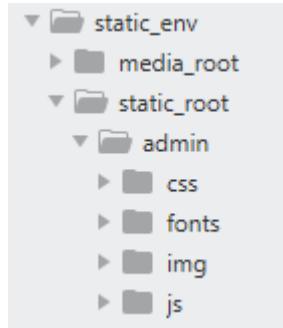
Si 'DEBUG = True' estamos en desarrollo

```
# SECURITY WARNING: don't run with debug turned on in production!  
DEBUG = True  
  
from django.contrib import admin  
from django.conf import setting  
from django.conf.urls.static import static  
from django.urls import path  
  
from boletin import views  
  
urlpatterns = [  
    path('admin/', admin.site.urls),  
    path('contact/', views.contact, name='contact'),  
    path('', views.inicio, name='inicio'),  
]  
  
if setting.DEBUG:  
    urlpatterns += static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)  
    urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Ejecutamos el comando “python manage.py collectstatic” para enviar nuestros archivos estáticos al servidor.

```
( proyecto) C:\Users\usuario\Desktop\proyecto\src>python manage.py collectstatic  
128 static files copied to 'C:\Users\usuario\Desktop\proyecto\static_env\static_root'.  
( proyecto) C:\Users\usuario\Desktop\proyecto\src>
```

Ahora podemos ver en la carpeta “static_root” se ha creado unas carpetas con fuentes, css... pero todavía no tenemos archivos estáticos.



23. Configuración Bootstrap

Bootstrap es un framework para diseño adaptable, cuando cambiemos el tamaño del navegador según el dispositivo, este se adapta perfectamente. Vamos a añadir un poco de diseño a nuestro proyecto.

Vamos a coger el código fuente de uno de los ejemplos de Bootstrap y añadirlo a nuestro proyecto. Creamos una nueva plantilla dentro de 'Templates' llamada 'base.html' y vamos a pegar todo el código ahí.

Top navbar Home Link Disabled

Search

Search

Navbar example

This example is a quick exercise to illustrate how the top-aligned navbar works. As you scroll, this navbar remains in its original position and moves with the rest of the page.

[View navbar docs »](#)

```

<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta name="description" content="">
    <meta name="author" content="Mark Otto, Jacob Thornton, and
    Bootstrap contributors">
    <meta name="generator" content="Hugo 0.88.1">
    <title>Top navbar example · Bootstrap v5.1</title>

    <link rel="canonical" href="https://getbootstrap.com/
docs/5.1/examples/navbar-static/">

    <!-- Bootstrap core CSS -->
    <link href="/docs/5.1/dist/css/bootstrap.min.css" rel="stylesheet"
integrity="sha384-1BmE4kWBq78iYhFlvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqy12QvZ6jIW3"
crossorigin="anonymous">

    <!-- Favicons -->
    <link rel="apple-touch-icon" href="/docs/5.1/assets/img/favicons/
apple-touch-icon.png" sizes="180x180">
    <link rel="icon" href="/docs/5.1/assets/img/favicons/favicon-32x32.png"
sizes="32x32" type="image/png">
    <link rel="icon" href="/docs/5.1/assets/img/favicons/favicon-16x16.png"
sizes="16x16" type="image/png">

```

En el fichero “view.py” estabamos renderizando “inicio.html”, y lo cambiamos por “base.html”. Volvemos a iniciar el servidor, abrimos nuestra página y se nos abrirá con el nuevo diseño.

En este caso vemos que está renderizando la plantilla pero no el css, para ello tendremos que añadir una hoja de estilo al proyecto.

```

    <!-- Bootstrap core CSS -->
    <link href="/docs/5.1/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-1BmE4kWBq78iYhFlvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqy12QvZ6jIW3" crossorigin="anonymous">

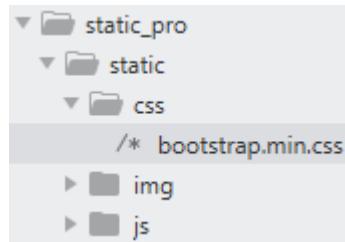
```

Lo abrimos y lo guardamos en src/static_pro/static/css

proyecto > src > static_pro > static > css

Nombre	Fecha de modifica
bootstrap.min	24/04/2022 13:57

Creamos dentro de static los directorios img y js



Tenemos que añadir una etiqueta "load static" para poder utilizar esa hoja de estilo dentro del fichero "base.html" y referenciamos nuestra carpeta.

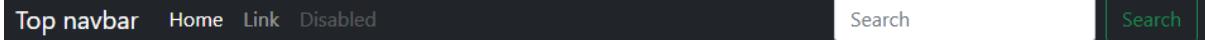
```
<!doctype html>
{% load static %}

<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta name="description" content="">
    <meta name="author" content="Mark Otto, Jacob Thornton, and
    Bootstrap contributors">
    <meta name="generator" content="Hugo 0.88.1">
    <title>Top navbar example · Bootstrap v5.1</title>

    <link rel="canonical" href="https://getbootstrap.com/
docs/5.1/examples/navbar-static/">

    <!-- Bootstrap core CSS -->
    <link href="{% static 'css/bootstrap.min.css' %}" rel="stylesheet"
integrity="sha384-1BmE4kWBq78iYhF1dvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqy12QvZ6jIW3"
crossorigin="anonymous">
```

Volvemos a recargar la página y veremos que se carga correctamente la página con sus estilos.



Navbar example

This example is a quick exercise to illustrate how the top-aligned navbar works. As you scroll, this navbar remains in its original position and moves with the rest of the page.

[View navbar docs »](#)

Como práctica de buenas maneras, y aunque no tengamos nada nuestro todavía subido, vamos a volver a escribir el comando para subir nuestros archivos estáticos.

```
( proyecto) C:\Users\usuario\Desktop\proyecto\src>python manage.py collectstatic
You have requested to collect static files at the destination
location as specified in your settings:
C:\Users\usuario\Desktop\proyecto\static_env\static_root

This will overwrite existing files!
Are you sure you want to do this?

Type 'yes' to continue, or 'no' to cancel: yes
2 static files copied to 'C:\Users\usuario\Desktop\proyecto\static_env\static_root', 128 unmodified.
(proyecto) C:\Users\usuario\Desktop\proyecto\src>
```

Volvemos a repetir la operación anterior para subir los archivos estáticos pero de JavaScript.

```
( proyecto) C:\Users\usuario\Desktop\proyecto\src>python manage.py collectstatic
You have requested to collect static files at the destination
location as specified in your settings:
C:\Users\usuario\Desktop\proyecto\static_env\static_root

This will overwrite existing files!
Are you sure you want to do this?

Type 'yes' to continue, or 'no' to cancel: yes
1 static file copied to 'C:\Users\usuario\Desktop\proyecto\static_env\static_root', 130 unmodified.
(proyecto) C:\Users\usuario\Desktop\proyecto\src>
```

24. Plantillas

Las plantillas son renderizadas en las vistas junto a la petición y al contexto. Si abrimos 'base.html' veremos que hay una barra de navegación, fundamental en estos casos. Lo que vamos a hacer es que las demás plantillas hereden este elemento para que se rendericen siempre sin repetir código.

Como ejemplo, creamos una plantilla nueva y pegamos la parte que queremos heredar.

```
<!--static navbar -->
<nav class="navbar navbar-expand-md navbar-dark bg-dark mb-4">
  <div class="container-fluid">
    <a class="navbar-brand" href="#">Top navbar</a>
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarCollapse" aria-controls="navbarCollapse" aria-expanded="false" aria-label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarCollapse">
      <ul class="navbar-nav me-auto mb-2 mb-md-0">
        <li class="nav-item">
          <a class="nav-link active" aria-current="page" href="#">Home</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="#">Link</a>
        </li>
        <li class="nav-item">
          <a class="nav-link disabled">Disabled</a>
        </li>
      </ul>
      <form class="d-flex">
        <input class="form-control me-2" type="search" placeholder="Search" aria-label="Search">
        <button class="btn btn-outline-success" type="submit">Search</button>
      </form>
    </div>
  </div>
</nav>
```

En el fichero "base.html" pondremos en el hueco vacío hacemos referencia al fichero "navbar.html".

```
{% include "navbar.html" %}
```

Una vez familiarizado con el código y cómo funciona la herencia entre plantillas, podemos seguir haciendo más ejemplos. Vamos a hacer lo mismo ahora pero con 'navbar example'. Nos llevamos la parte de la barra a 'inicio.html' y con 'extends' aplicamos la herencia.

```
base.html      x    inicio.html      x    navbar.html      x    views.py  
  
{% extends "base.html" %}  
<div class="bg-light p-5 rounded">  
  <h1>Navbar example</h1>  
  <p class="lead">This example is a quick exercise to illustrate how  
  the top-aligned navbar works. As you scroll, this navbar remains in  
  its original position and moves with the rest of the page.</p>  
  <a class="btn btn-lg btn-primary" href="/docs/5.1/components/navbar/"  
     role="button">View navbar docs &raquo;</a>  
</div>  
  
<!--<h1>Hola mundo</h1> -->  
{% titulo %}  
<br/>  
{% request.user %}  
<hr/>  
<br/>  
  
<form method="POST" action="">{% csrf_token %}  
{{ el_form.as_p }}  
<input type='submit' value='Registrate' />  
</form>
```

```
base.html      x    inicio.html      ●    navbar.html      x    views.py  
  
    <!-- Custom styles for this template -->  
    <link href="{% static 'navbar-top.css' %}" rel="stylesheet">  
</head>  
<body>  
  
{% include "navbar.html" %}  
  
<main class="container">  
  
{% block content %}  
{% endblock %}  
  
</main>  
  
    <script src="/docs/5.1/dist/js/bootstrap.bundle.min.js" integrity="sha384-ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENB0LRn5q+8nbTov4+1p  
    " crossorigin="anonymous"></script>  
  
    </body>  
</html>
```

Para que aparezca el formulario, basta con aplicar el 'block content' en la parte correspondiente:

```
base.html           inicio.html           navbar.html           views.py

{% extends "base.html" %}



<h1>Navbar example</h1>
    <p class="lead">This example is a quick exercise to illustrate how
        the top-aligned navbar works. As you scroll, this navbar remains in
        its original position and moves with the rest of the page.</p>
    <a class="btn btn-lg btn-primary" href="/docs/5.1/components/navbar/">
        View navbar docs &raquo;
    </a>
</div>

<!--<h1>Hola mundo</h1> -->
{{ titulo }}
<br/>
{{ request.user }}
<hr/>
<br/>

{% block content %}
<form method="POST" action="">{{ csrf_token }}
{{ el_form.as_p }}
<input type='submit' value='Registrate' />
</form>
{% endblock %}


```

Nombre:

Email:

Todo lo que queremos renderizar de 'inicio.html' tenemos que meterlo en un 'block content' en la 'base.html'. Hacemos lo mismo con 'jumbotron' y ya nos quedará el diseño aplicado a nuestro formulario.

```
base.html      x  inicio.html      x  navbar.html      x  views.py      x
{% extends "base.html" %}

{% block jumbotron %}


# Navbar example



This example is a quick exercise to illustrate how the top-aligned navbar works. As you scroll, this navbar remains in its original position and moves with the rest of the page.

View navbar docs &raquo;


{% endblock %}

{{ titulo }}
<br/>
{{ request.user }}
<hr/>
<br/>

{% block content %}
<form method="POST" action="">{{ csrf_token }}
{{ el_form.as_p }}
<input type='submit' value='Registrate' />
</form>
{% endblock %}
```

PROBAR DJANGO Home Link Disabled

Search

Search

Navbar example

This example is a quick exercise to illustrate how the top-aligned navbar works. As you scroll, this navbar remains in its original position and moves with the rest of the page.

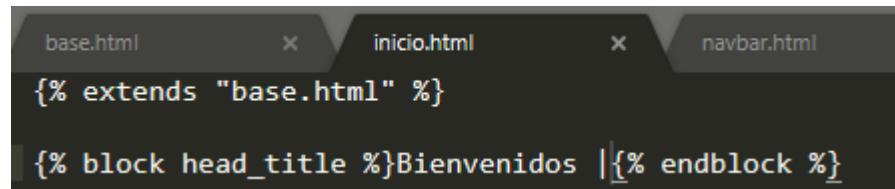
[View navbar docs »](#)

Nombre:

Email:

Si queremos que aparezca algo dentro de un bloque de donde heredamos, podemos hacer lo siguiente:

```
<title>{% block head_title %}{% endblock %}Proyecto</title>
```



```

base.html      x  inicio.html      x  navbar.html
{% extends "base.html" %}

| {% block head_title %}Bienvenidos |{% endblock %}

```

Vamos a seguir limpiando 'base.html', vamos a crear una nueva plantilla 'head_css.html' y pegaremos:



```

base.html x  javascript.html x  head_css.html x  inicio.html x  navbar.html x  views.

{% load static %}
<!-- Bootstrap core CSS --&gt;
&lt;link href="{% static 'css/bootstrap.min.css' %}" rel="stylesheet" integrity="sha384-1BmE4kWBq78iYhFlvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqy12QvZ6jIW3" crossorigin="anonymous"&gt;
<!-- Favicons --&gt;
&lt;link rel="apple-touch-icon" href="/docs/5.1/assets/img/favicons/apple-touch-icon.png" sizes="180x180"&gt;
&lt;link rel="icon" href="/docs/5.1/assets/img/favicons/favicon-32x32.png" sizes="32x32" type="image/png"&gt;
&lt;link rel="icon" href="/docs/5.1/assets/img/favicons/favicon-16x16.png" sizes="16x16" type="image/png"&gt;
&lt;link rel="manifest" href="/docs/5.1/assets/img/favicons/manifest.json"&gt;
&lt;link rel="mask-icon" href="/docs/5.1/assets/img/favicons/safari-pinned-tab.svg" color="#7952b3"&gt;
&lt;link rel="icon" href="/docs/5.1/assets/img/favicons/favicon.ico"&gt;
&lt;meta name="theme-color" content="#7952b3"&gt;

&lt;style&gt;
.bd-placeholder-img {
  font-size: 1.125rem;
  text-anchor: middle;
  -webkit-user-select: none;
  -moz-user-select: none;
  user-select: none;
}

@media (min-width: 768px) {
  .bd-placeholder-img-lg {
    font-size: 3.5rem;
  }
}
&lt;/style&gt;

<!-- Custom styles for this template --&gt;
&lt;link href="{% static 'navbar-top.css' %}" rel="stylesheet"&gt;
</pre>

```

Continuamos limpiando el fichero base.html quedando así:

```

base.html  x  javascript.html  x  head_css.html  x  inicio.html  x  navbar.html  x  views.py
<!doctype html>
{%
  load static %
}
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta name="description" content="">
    <meta name="author" content="Mark Otto, Jacob Thornton, and
      Bootstrap contributors">
    <meta name="generator" content="Hugo 0.88.1">
    <title>{% block head_title %}{% endblock %}Proyecto</title>

    <link rel="canonical" href="https://getbootstrap.com/
      docs/5.1/examples/navbar-static/">

    {% include "head_css.html" %}

  </head>
  <body>

    {% include "navbar.html" %}

    <main class="container">

      {% block jumbotron %}
      <div class="bg-light p-5 rounded">
        {% block jumbotron_content %}

        {% endblock %}
        </div>
      {% endblock %}

      {% block content %}
      {% endblock %}
    </main>

    {% include "javascript.html" %}
    </body>
  </html>

```

Una vez terminado recargamos la página y tendremos lo siguiente:

The screenshot shows a web browser displaying a Django application. At the top is a dark navigation bar with the text "PROBAR DJANGO" and links for "Home", "Link", and "Disabled". To the right of the navigation bar is a search bar with a "Search" button. The main content area has a light gray background. At the top of this area is a top-aligned navbar with the word "INICIO" in large, bold, black capital letters. Below the navbar, there is explanatory text: "This example is a quick exercise to illustrate how the top-aligned navbar works. As you scroll, this navbar remains in its original position and moves with the rest of the page." Below this text is a blue rectangular button with white text that says "View navbar docs ». Further down, there is a registration form with fields for "Nombre:" and "Email:", each accompanied by a text input box. Below these fields is a "Regístrate" button.

25. Django Crispy Forms

Es una aplicación o un paquete de Django para añadir o mejorar el aspecto de nuestro formulario sin escribir mucho código, siguiendo el método DRY (Dont Repeat Yourself). Primero habrá que instalarlo:

```
( proyecto ) C:\Users\usuario\Desktop\proyecto\src>pip install --upgrade django-crispy-forms
Collecting django-crispy-forms
  Downloading django_crispy_forms-1.14.0-py3-none-any.whl (133 kB)
    133.3/133.3 KB 783.9 kB/s eta 0:00:00
Installing collected packages: django-crispy-forms
Successfully installed django-crispy-forms-1.14.0
( proyecto ) C:\Users\usuario\Desktop\proyecto\src>
```

Después habrá que agregarlo a nuestras 'INSTALLED_APPS' dentro de 'settings.py'.

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'crispy_forms',
    'boletin'
]
```

Hacemos las migraciones.

```
( proyecto ) C:\Users\usuario\Desktop\proyecto\src>python manage.py makemigrations
No changes detected

( proyecto ) C:\Users\usuario\Desktop\proyecto\src>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, boletin, contenttypes, sessions
Running migrations:
  No migrations to apply.

( proyecto ) C:\Users\usuario\Desktop\proyecto\src>
```

Agregamos la línea 'template pack' en 'settings.py'

```
CRISPY_TEMPLATE_PACK = 'bootstrap3'
```

Abrimos 'inicio.html', lo cargamos:

```
inicio.html      x      settings.py      x

{% extends "base.html" %}
{% load crispy_forms_tags %}

{% block head_title %}Bienvenidos |{% endblock %}

{% block jumbotron_content %}
    <h1>INICIO</h1>
    <p class="lead">This example is a quick exercise to illustrate how
    the top-aligned navbar works. As you scroll, this navbar remains in
    its original position and moves with the rest of the page.</p>
    <a class="btn btn-lg btn-primary" href="/docs/5.1/components/navbar/"
        role="button">View navbar docs &raquo;</a>
    </div>
{% endblock %}

{{ titulo }}
<br/>
{{ request.user }}
<hr/>
<br/>

{% block content %}
<form method="POST" action="">{{ csrf_token }}
{{ el_form|crispy }}
<input type='submit' value='Registrate' />
</form>
{% endblock %}
```

Y en vez de {{ el_form.as_p }} escribimos {{ el_form|crispy }}, y se verá algo tal que así:

PROBAR DJANGO Home Link Disabled

Search

Search

INICIO

This example is a quick exercise to illustrate how the top-aligned navbar works. As you scroll, this navbar remains in its original position and moves with the rest of the page.

[View navbar docs »](#)

Nombre

Email*

26. Estilo: Bootstrap 1

Continuamos modificando el estilo de la página usando Bootstrap. Si queremos que el “jumbotron” se extienda a lo largo de toda la página, añadiremos “-fluid” en el fichero “base.html”

```
<main class="container-fluid">

    {% block jumbotron %}
        <div class="bg-light p-5 rounded">
            {% block jumbotron_content %}

                {% endblock %}
                </div>
            {% endblock %}

            {% block content %}
            {% endblock %}
        </main>

        {% include "javascript.html" %}
        </body>
    </html>
```

PROBAR DJANGO Home Link Disabled

Search

Search

INICIO

This example is a quick exercise to illustrate how the top-aligned navbar works. As you scroll, this navbar remains in its original position and moves with the rest of the page.

[View navbar docs »](#)

Nombre

Email*

Vamos a hacer cosas más importantes, por ejemplo:

```
inicio.html      x  base.html      x  navbar.html      x  settings.py      x
{% extends "base.html" %}
{% load crispy_forms_tags %}

{% block head_title %}Bienvenidos |{% endblock %}

{% block jumbotron_content %}
<h1>Probar Django</h1>
<p class="lead">Un proyecto para principiantes. El objetivo es construir una página web simple a la par que elegante en muy poco tiempo</p>
<a class="btn btn-lg btn-primary" href="/docs/5.1/components/navbar/" role="button">Únete &raquo;</a>
</div>
{% endblock %}

{{ titulo }}
<br/>
{{ request.user }}
<hr/>
<br/>

{% block content %}
<form method="POST" action="">{% csrf_token %}
{{ el_form|crispy }}
<input type='submit' value='Regístrate' />
</form>
<hr/>
{% endblock %}
```

PROBAR DJANGO Home Link Disabled

Search

Search

Probar Django

Un proyecto para principiantes. El objetivo es construir una página web simple a la par que elegante en muy poco tiempo

[Únete »](#)

Nombre

Email*

Vamos a insertar un vídeo a la derecha de nuestra página. Para empezar a hacer eso, donde tenemos nuestro contenido del 'jumbotron', agregamos lo siguiente:

```
inicio.html           x     base.html           x     navbar.html           x     settings.py  
  
{% extends "base.html" %}  
{% load crispy_forms_tags %}  
  
{% block head_title %}Bienvenidos |{% endblock %}  
  
{% block jumbotron_content %}  
<div class="row">  
    <div class="col-sm-6">  
        <h1>Probar Django</h1>  
        <p class="lead">Un proyecto para principiantes. El objetivo es  
        construir una página web simple a la par que elegante en muy poco  
        tiempo</p>  
        <a class="btn btn-lg btn-primary" href="/docs/5.1/components/navbar/"  
            role="button">Únete &raquo;</a>  
</div>  
    </div>  
{% endblock %}  
  
{{ titulo }}  
<br/>  
{{ request.user }}  
<hr/>  
<br/>  
  
{% block content %}  
<form method="POST" action="">{% csrf_token %}  
{{ el_form|crispy }}  
<input type='submit' value='Regístrate' />  
</form>  
<hr/>
```

PROBAR DJANGO Home Link Disabled

Probar Django

Un proyecto para principiantes. El objetivo es
construir una página web simple a la par que
elegante en muy poco tiempo

[Únete »](#)

Nombre

Email*

Podemos agregar un recuadro negro donde irá nuestro video:

```
<div class="col-sm-6" style="background-color:black;height:300px;"></div>
```

PROBAR DJANGO Home Link Disabled

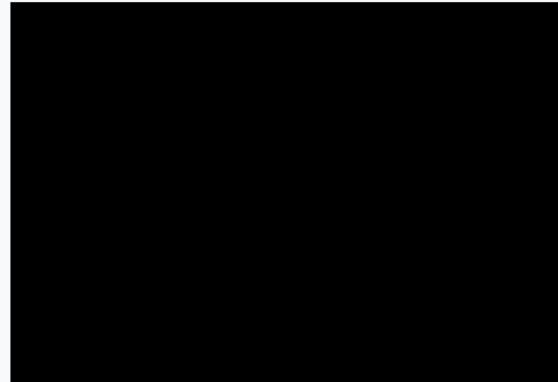
Search

Search

Probar Django

Un proyecto para principiantes. El objetivo es construir una página web simple a la par que elegante en muy poco tiempo

Únete »



Nombre

Email*

[Regístrate](#)

Modificar parte del formulario.

```
inicio.html      x     views.py      x     base.html      x     navbar.html      x
{{ request.user }}

<hr/>
<br/>

{% block content %}


<div class="col-sm-3 col-xs-12 pull-right">
        <p class="lead">{{ titulo }}</p>
    <form method="POST" action="">{{ csrf_token }}
    {{ el_form|crispy }}
    <input type='submit' value='Registrate' />
    </form>
</div>
<div class="col-sm-3">
    <p class="lead">Creado con Django & Bootstrap</p>
</div>
<div class="col-sm-3">
    <p class="lead">Y con mucho amor, claro</p>
</div>
<div class="col-sm-3">
    <p class="lead">Código abierto, siempre</p>
</div>
<br/>
{% endblock %}


```



Probar Django

Un proyecto para principiantes. El objetivo es construir una página web simple a la par que elegante en muy poco tiempo

[Únete »](#)

Bienvenido Fran

Creado con Django & Bootstrap

Y con mucho amor, claro

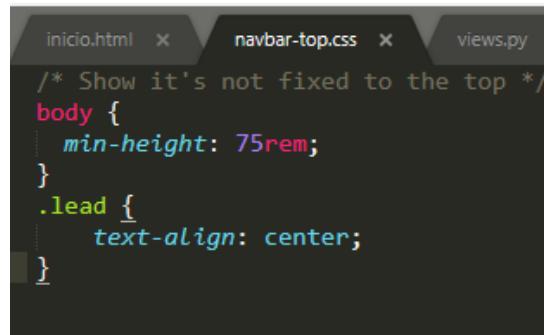
Código abierto, siempre

Nombre

Email*

27. Estilo: CSS Custom

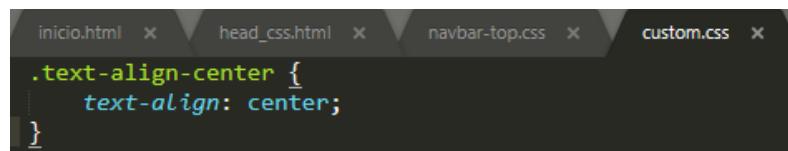
Vamos a crear una hoja de estilo nueva para cambiar la alineación del texto. Vamos a nuestro código, y en el css escribimos:



```
/* Show it's not fixed to the top */
body {
    min-height: 75rem;
}
.lead {
    text-align: center;
}
```

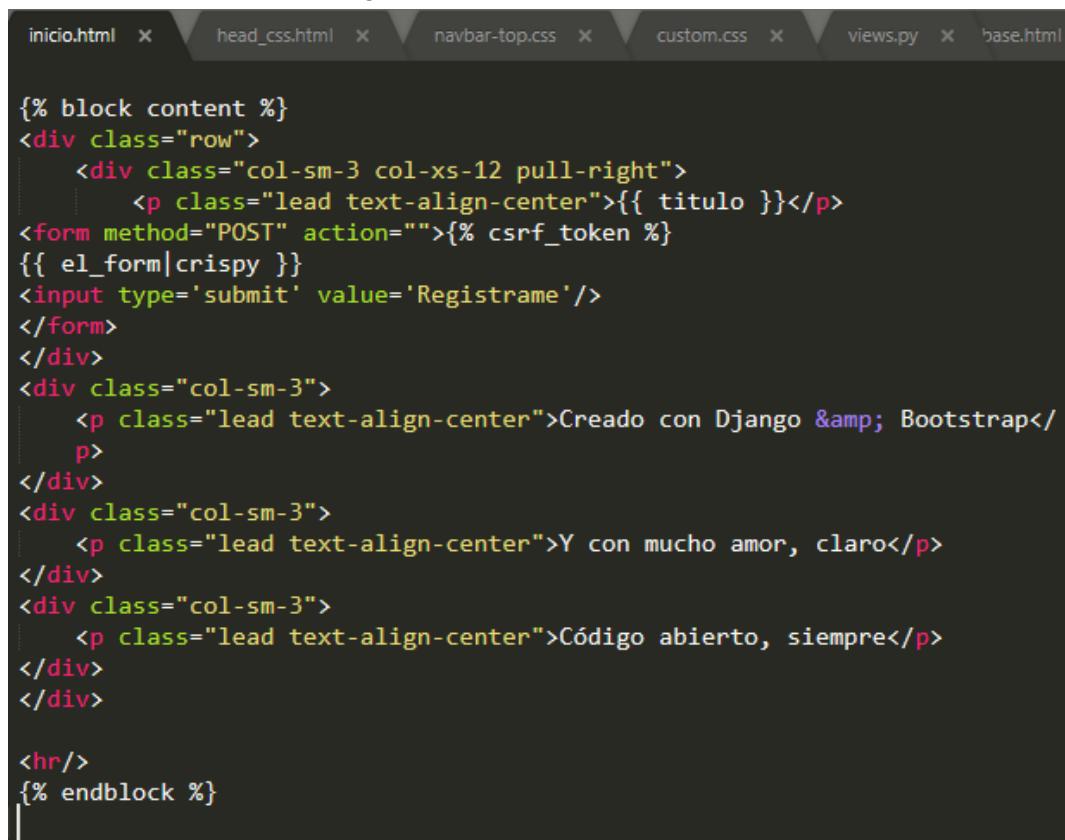
Así nuestro texto quedará alineado al centro. La cosa de hacerlo de esta forma es que si tenemos otro texto de clase 'lead' y se nos olvidará, tendríamos problemas. Lo mejor es hacerlo de la siguiente manera.

Vamos a crear una hoja de estilo nueva 'custom.css' y crear una clase nueva que vamos a aplicar a nuestra etiqueta <p>



```
.text-align-center {
    text-align: center;
}
```

Y a todas las etiquetas <p> lo pegamos.



```
{% block content %}


<p class="lead text-align-center">{{ titulo }}</p>


    <form method="POST" action="">{{ csrf_token %}}
    {{ el_form|crispy }}
    <input type='submit' value='Registrate' />
    </form>



<p class="lead text-align-center">Creado con Django & Bootstrap</p>



<p class="lead text-align-center">Y con mucho amor, claro</p>



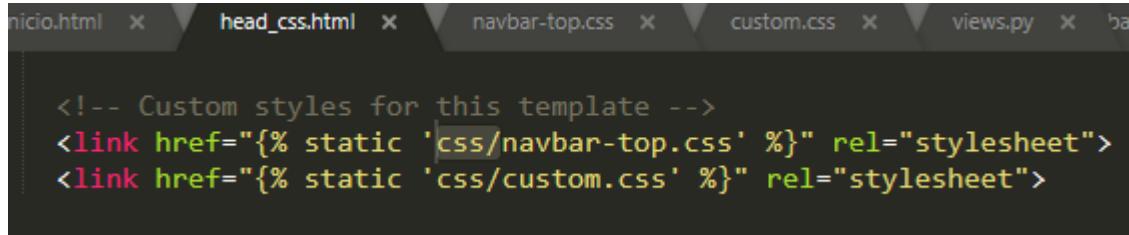
<p class="lead text-align-center">Código abierto, siempre</p>



---


{% endblock %}
```

Por último, tenemos que ir a 'head_css.html' y crear una ruta para nuestro 'custom.css'



A screenshot of a code editor showing a file with the following content:

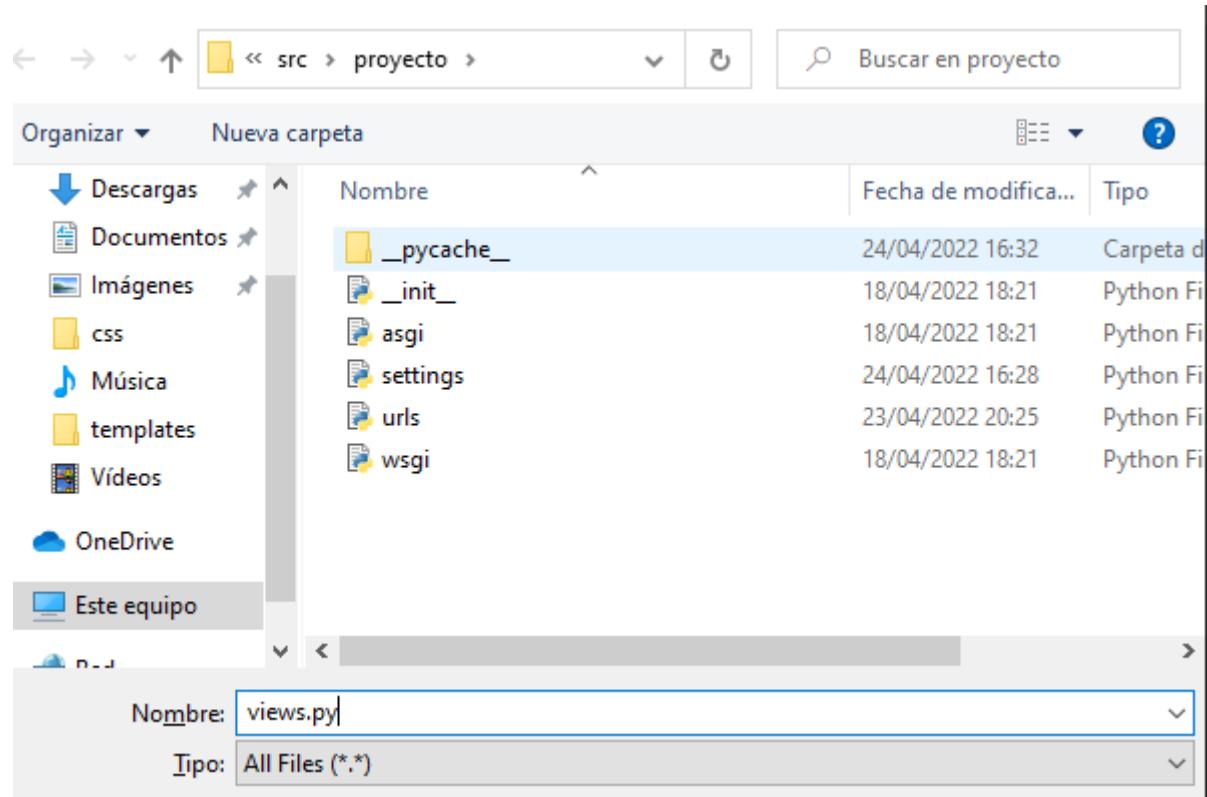
```
<!-- Custom styles for this template -->
<link href="{% static 'css/navbar-top.css' %}" rel="stylesheet">
<link href="{% static 'css/custom.css' %}" rel="stylesheet">
```

Asegurarse siempre que agreguemos nuevos archivos estáticos de hacer el 'collectstatic' en cmd.

```
( proyecto ) C:\Users\usuario\Desktop\proyecto\src>python manage.py collectstatic
You have requested to collect static files at the destination
location as specified in your settings:
C:\Users\usuario\Desktop\proyecto\static_env\static_root
This will overwrite existing files!
Are you sure you want to do this?
Type 'yes' to continue, or 'no' to cancel: yes
2 static files copied to 'C:\Users\usuario\Desktop\proyecto\static_env\static_root', 130 unmodified.
( proyecto ) C:\Users\usuario\Desktop\proyecto\src>
```

28. Enlaces con nombres URL

Vamos a configurar nuestra barra de navegación para que podamos movernos entre ella.
Vamos a crear una vista fuera del Django:



Vamos a hacer una importación en 'urls.py'

```
views.py — proyecto x urls.py x views.py — boletin x navbar.html x
from django.contrib import admin
from django.conf import settings
from django.urls.static import static
from django.urls import path

from boletin import views
from .views import about

urlpatterns = [
    path('admin/', admin.site.urls),
    path('contact/', views.contact, name='contact'),
    path('', views.inicio, name='inicio'),
    path('about/', about, name='about'),
]

if settings.DEBUG:
    urlpatterns += static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)
    urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Y en 'navbar.html' vamos a escribir las rutas a fuego para que se enlacen con el navegador (esto nos vale para que cuando se cambie el nombre del navegador, la ruta no varie)

```
views.py — proyecto      navbar.html      urls.py      views.py — boletin
<!--static navbar -->
<nav class="navbar navbar-expand-md navbar-dark bg-dark mb-4">
  <div class="container">
    <a class="navbar-brand" href="/">PROBAR DJANGO</a>
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarCollapse" aria-controls="navbarCollapse" aria-expanded="false" aria-label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarCollapse">
      <ul class="navbar-nav me-auto mb-2 mb-md-0">
        <li>
          <a href="/">Home</a></li>
        <li><a href="{% url 'about' %}">About</a></li>
        <li><a href="{% url 'contact' %}">Contact</a></li>
      </ul>
      <form class="d-flex">
        <input class="form-control me-2" type="search" placeholder="Search" aria-label="Search">
        <button class="btn btn-outline-success" type="submit">Search</button>
      </form>
    </div>
  </div>
</nav>
```

Por ejemplo si pinchamos en 'Contact' se nos abrirá el html sin ningún estilo.

The screenshot shows a web browser window with the following details:

- URL Bar:** 127.0.0.1:8000/contact/
- Page Content:**
 - A text input field labeled "Nombre:" followed by an empty input box.
 - A text input field labeled "Email:" followed by an empty input box.
 - A large text area labeled "Mensaje:" followed by an empty text area.
 - A blue "Registerme" button at the bottom.

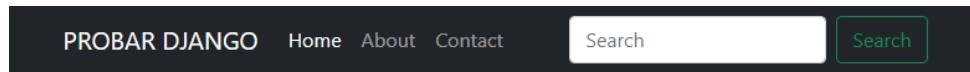
29. Estilo: Bootstrap 2

Cambiemos el estilo de 'Contact', abrimos 'forms.html' y vamos a insertar 'crispy' y vamos a borrar el 'jumbotron' en nuestra página de Contacto, por lo cual vamos a añadir unas etiquetas <dir> dentro del 'block content'

```
inicio.html      x      forms.html      x      views.py      x      base.html      x
{% extends "base.html" %} 
{% load crispy_forms_tags %}

{{ titulo }} 
<hr/>

{% block content %}
<div class="row">
    <div class="col-sm-6 col-sm-offset-3">
        {% if titulo %}
            <h1 class="text-align-center">{{ titulo }}</h1>
        {% endif %}
        <form method="POST" action="">{{ csrf_token }}
            {{ form|crispy }}
            <input class="btn btn-primary" type='submit' value='Enviar' />
        </form>
    </div>
</div>
{% endblock %}
```



Contacto

Nombre

Email*

Mensaje*

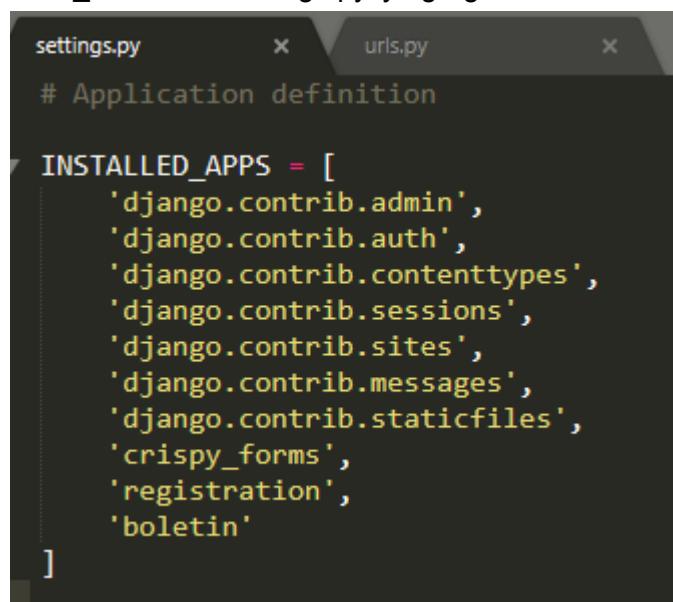
30. Django Registration Redux

A continuación vamos a instalar Django Registration Redux, para ello escribimos lo siguiente en la cmd:

```
( proyecto ) C:\Users\usuario\Desktop\proyecto>pip install django-registration-redux
Collecting django-registration-redux
  Downloading django_registration_redux-2.10-py2.py3-none-any.whl (213 kB)
    213.2/213.2 KB 1.6 MB/s eta 0:00:00
Installing collected packages: django-registration-redux
Successfully installed django-registration-redux-2.10

( proyecto ) C:\Users\usuario\Desktop\proyecto>
```

Vamos a las 'INSTALLED_APPS' de 'settings.py' y agregamos lo nuevo.



```
settings.py      x      urls.py      x
# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'crispy_forms',
    'registration',
    'boletin'
]
```

Además, tenemos que añadir unas líneas:

```
ACCOUNT_ACTIVATION_DAYS = 7
REGISTRATION_AUTO_LOGIN = True
CRISPY_TEMPLATE_PACK = 'bootstrap3'
```

Cambiar la siguiente línea:

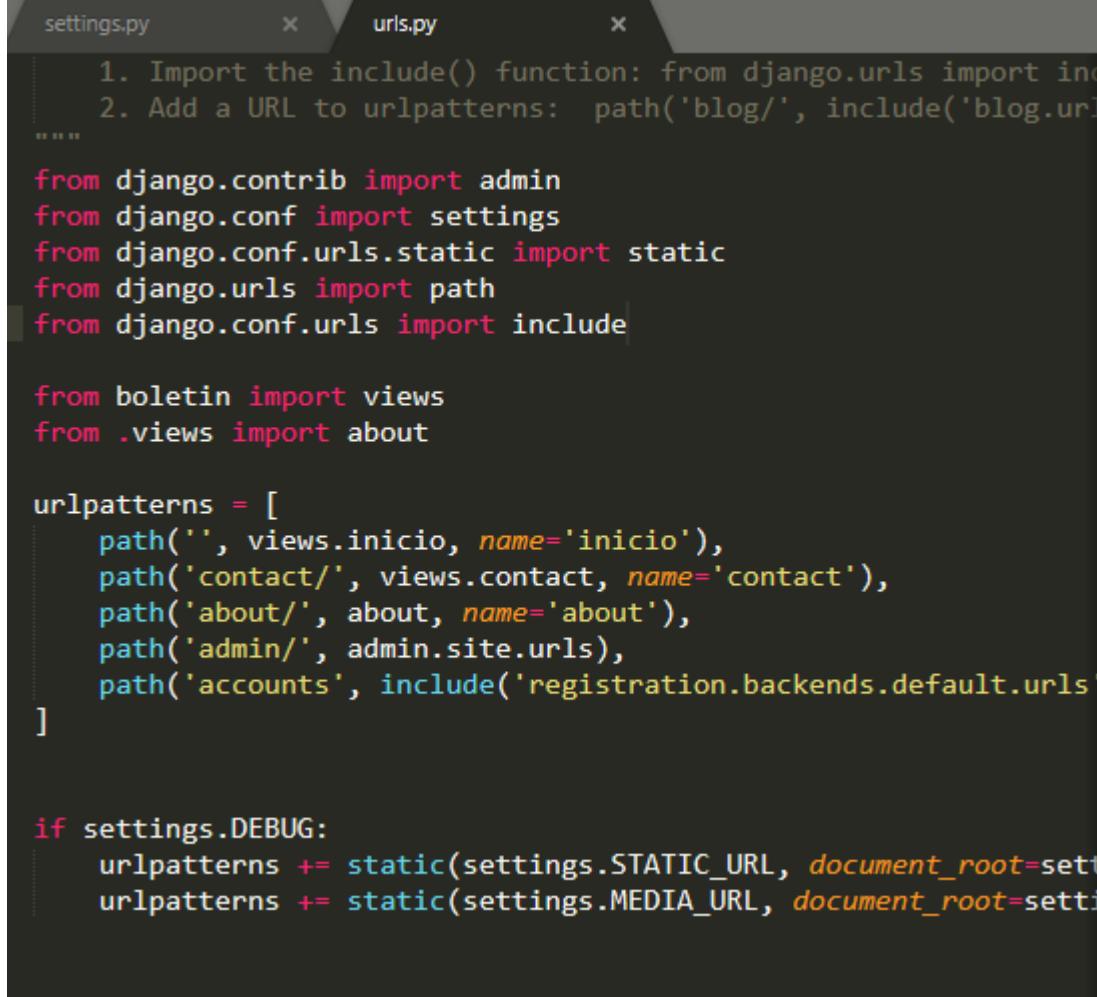
```
STATIC_URL = '/static/'
MEDIA_URL = '/media/'
```

Y hacemos las migraciones.

```
( proyecto ) C:\Users\usuario\Desktop\proyecto\src>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, boletin, contenttypes, registration, sessions, sites
Running migrations:
  Applying registration.0001_initial... OK
  Applying registration.0002_registrationprofile_activated... OK
  Applying registration.0003_migrate_activatedstatus... OK
  Applying registration.0004_supervisedregistrationprofile... OK
  Applying registration.0005_activation_key_sha256... OK
  Applying sites.0001_initial... OK
  Applying sites.0002_alter_domain_unique... OK
```

```
( proyecto ) C:\Users\usuario\Desktop\proyecto\src>
```

Posteriormente configuraremos el fichero "urls.py".



```
settings.py      x     urls.py      x

1. Import the include() function: from django.urls import include
2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))
"""

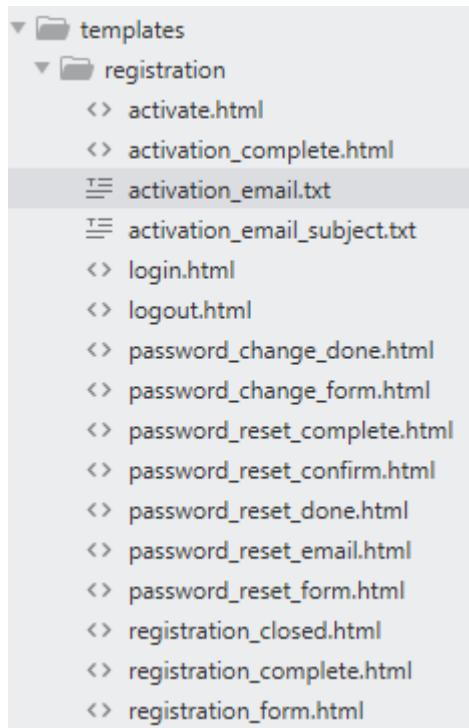
from django.contrib import admin
from django.conf import settings
from django.conf.urls.static import static
from django.urls import path
from django.conf.urls import include

from boletin import views
from .views import about

urlpatterns = [
    path('', views.inicio, name='inicio'),
    path('contact/', views.contact, name='contact'),
    path('about/', about, name='about'),
    path('admin/', admin.site.urls),
    path('accounts', include('registration.backends.default.urls'))
]

if settings.DEBUG:
    urlpatterns += static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)
    urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Tenemos que descargarnos las plantillas de 'registration': ([página](#))



Y si nos vamos a 'accounts/register' nos parecerá el formulario de registro

PROBAR DJANGO Home About Contact

Registrarte Gratis!

Nombre de usuario*

ReCompleta este campo teres como máximo.
Únicamente letras, dígitos y @//+/-/_

Correo Electrónico*

Contraseña*

• Su contraseña no puede asemejarse tanto a su otra información personal.
• Su contraseña debe contener al menos 8 caracteres.
• Su contraseña no puede ser una clave utilizada comúnmente.
• Su contraseña no puede ser completamente numérica.

Contraseña (confirmación)*

Para verificar, introduzca la misma contraseña anterior.

Activar Windows
Ve a Configuració
Windows.

Ya tienes cuenta? [Iniciar Sesión.](#)

A screenshot of a code editor showing a single file named `registration_form.html`. The file contains Django template code. At the top, there are tabs for `settings.py`, `registration_form.html`, and `urls.py`. The code itself starts with extending the `base.html` template and loading i18n and crispy_forms_tags. It then defines a content block with a centered strong message "Registrarte Gratis!". Inside this block, there's a form for registration with CSRF token and crispy form fields. After the form, there's a large button with the translated text "Registrarme". Below this, there's another row with a column for login links. The template ends with an `endblock` tag.

```
{% extends "base.html" %}

{% load i18n %}
{% load crispy_forms_tags %}

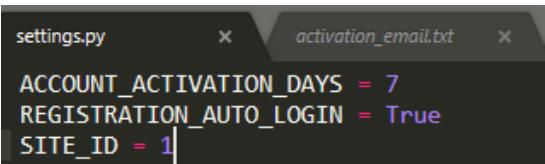
{% block content %}
<div class='row'>
<div class='col-sm-6 col-sm-offset-3'>
    <h2 class='text-align-center'><strong>Registrarte Gratis!</strong></h2><br/>
<form method="post" action=".">
    {% csrf_token %}
    {{ form|crispy }}

    <input class='btn btn-block btn-primary' type="submit" value="{% trans 'Registrarme' %}" />
</form>
</div>
</div>

<hr/>
<div class='row'>
<div class='col-sm-6 col-sm-offset-3 text-align-center'>
<p>Ya tienes cuenta? <a href="{% url 'auth_login' %}"> Iniciar Sesión.</a></p>
</div>
</div>
{% endblock %}
```

31. Django Registration Redux 2

Tendremos que añadir "SITE_ID = 1" en el fichero "settings.py", para que nos funcione "accounts/login/".



settings.py

```
ACCOUNT_ACTIVATION_DAYS = 7
REGISTRATION_AUTO_LOGIN = True
SITE_ID = 1
```

activation_email.txt

127.0.0.1:8000/accounts/login/

PROBAR DJANGO Home About Contact Search Search

Iniciar Sesión

Nombre de usuario*

Contraseña*

[Iniciar sesión](#)

Has olvidado tu contraseña? [Restablecer!](#)

No tienes cuenta? [Registrarte!](#)

'SITE_ID' tiene que ver con una aplicación de Django que no esta activada por defecto. Si entramos ahora en admin, lo veremos.

Sitio administrativo

AUTENTICACIÓN Y AUTORIZACIÓN

Grupos	Añadir	Modificar
Usuarios	Añadir	Modificar

BOLETIN

Registrados	Añadir	Modificar
-------------	------------------------	---------------------------

REGISTRATION

Perfiles de registro	Añadir	Modificar
----------------------	------------------------	---------------------------

SITIOS

Sitios	Añadir	Modificar
--------	------------------------	---------------------------

Seleccione sitio a modificar

A screenshot of a web-based administration interface. At the top, there is a search bar with a magnifying glass icon and a 'Buscar' button. Below the search bar, there is a dropdown menu labeled 'Acción:' with a '-----' option, and a 'Ir' button next to it. To the right of these, it says 'seleccionados 0 de 1'. The main area contains a table with two columns: 'NOMBRE DE DOMINIO' and 'NOMBRE A MOSTRAR'. There is one row in the table with a checkbox next to 'example.com' and 'example.com' in the 'NOMBRE A MOSTRAR' column. At the bottom left, it says '1 sitio'.

Si intentamos crear un nuevo usuario nos dará error:

SMTPAuthenticationError at /accounts/register/

(535, b'5.7.8 Username and Password not accepted. Learn more at\n5.7.8

https://support.google.com/mail/?p=BadCredentials q16-

20020adff950000000b00205aa05fa03sm7868368wrr.58 - gsmtp')

Request Method: POST

Request URL: http://127.0.0.1:8000/accounts/register/

Django Version: 4.0.4

Exception Type: SMTPAuthenticationError

Exception Value: (535, b'5.7.8 Username and Password not accepted. Learn more at\n5.7.8 https://support.google.com/mail/?p=BadCredentials q16-20020adff950000000b00205aa05fa03sm7868368wrr.58 - gsmtp')

Exception Location: C:\Users\usuario\AppData\Local\Programs\Python\Python310\lib\smtplib.py, line 662, in auth

Python Executable: C:\Users\usuario\Desktop\proyecto\Scripts\python.exe

Python Version: 3.10.4

Python Path: ['C:\\Users\\usuario\\Desktop\\\\proyecto\\\\src', 'C:\\Users\\usuario\\AppData\\\\Local\\\\Programs\\\\Python\\\\Python310\\\\python310.zip', 'C:\\Users\\usuario\\AppData\\\\Local\\\\Programs\\\\Python\\\\Python310\\\\DLLs', 'C:\\Users\\usuario\\AppData\\\\Local\\\\Programs\\\\Python\\\\Python310\\\\lib', 'C:\\Users\\usuario\\AppData\\\\Local\\\\Programs\\\\Python\\\\Python310', 'C:\\Users\\usuario\\Desktop\\\\proyecto', 'C:\\Users\\usuario\\Desktop\\\\proyecto\\\\lib\\\\site-packages']

Server time: Mon, 25 Apr 2022 06:23:12 +0000

Traceback [Switch to copy-and-paste view](#)

C:\Users\usuario\Desktop\proyecto\lib\site-packages\django\core\handlers\exception.py, line 55, in inner

55. response = get_response(request)

► Local vars

C:\Users\usuario\Desktop\proyecto\lib\site-packages\django\core\handlers\base.py, line 197, in _get_response

197. response = wrapped_callback(request, *callback_args, **callback_kwargs)

► Local vars

C:\Users\usuario\Desktop\proyecto\lib\site-packages\django\views\generic\base.py, line 84, in view

84. return self.dispatch(request, *args, **kwargs)

► Local vars

C:\Users\usuario\Desktop\proyecto\lib\site-packages\django\utils\decorators.py, line 46, in _wrapper

46. return bound_method(*args, **kwargs)

► Local vars

Activar Window

Ve a Configuración p
Windows.

Pero si nos vamos a perfiles de registro veremos que se ha creado pero la cuenta no está activada.

Seleccione perfil de registro a modificar

AÑADIR P

Buscar

Acción: ----- Ir seleccionados 0 de 1

<input type="checkbox"/> USUARIO	ACTIVATION KEY EXPIRED
<input type="checkbox"/> abc	False

1 perfil de registro

Para activarla tendremos que poner lo siguiente en el fichero “activation_email.txt”:

```
settings.py      activation_email.txt      urls.py
{% load i18n %}

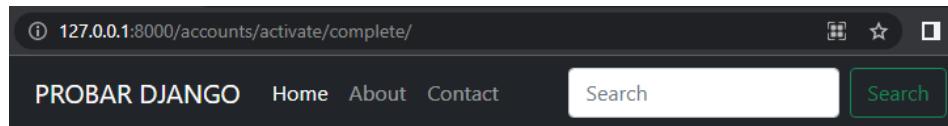
{% trans "Activa tu cuenta en" %} {{ site.name }}:

Hola,
Haz click en el enlace para activar tu cuenta.
http://{{ site.domain }}{{ url 'registration_activate' activation_key }}
[{{ blocktrans }}Enlace válido durante {{ expiration_days }} días.{% endblocktrans %}

-Team CFE
```

Nos vamos a la página y escribimos lo siguiente

<http://127.0.0.1:8000/accounts/activate/de3dbff662bd8323b954420538da9a7bc52cb1697075432ecd7bda3036decaf1>



Y ahora al entrar en perfil de usuario vemos que está activada

Modificar perfil de registro

Registration information for abc

Usuario:

Clave de activación:

Activated

Al intentar iniciar sesión nos da error.

Page not found (404)

Request Method: GET
Request URL: http://127.0.0.1:8000/accounts/profile/

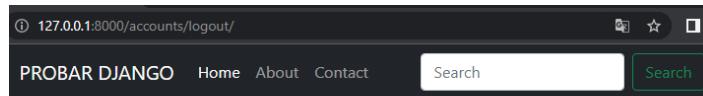
Using the URLconf defined in `proyecto.urls`, Django tried these URL patterns, in this order:

1. admin/
2. [name='inicio']
3. contact/ [name='contact']
4. about/ [name='about']
5. accounts/ activate/complete/ [name='registration_activation_complete']
6. accounts/ activate/resend/ [name='registration_resend_activation']
7. accounts/ activate/<activation_key>/ [name='registration_activate']
8. accounts/ register/complete/ [name='registration_complete']
9. accounts/ register/closed/ [name='registration_disallowed']
10. accounts/ register/ [name='registration_register']
11. accounts/ login/ [name='auth_login']
12. accounts/ logout/ [name='auth_logout']
13. accounts/ password/change/ [name='auth_password_change']
14. accounts/ password/change/done/ [name='auth_password_change_done']
15. accounts/ password/reset/ [name='auth_password_reset']
16. accounts/ password/reset/complete/ [name='auth_password_reset_complete']
17. accounts/ password/reset/done/ [name='auth_password_reset_done']
18. accounts/ password/reset/confirm/<uidb64>/<token>/ [name='auth_password_reset_confirm']
19. static(/P<path>.)\$
20. ^media/(?P<path>.)\$

The current path, `accounts/profile/`, didn't match any of these.

You're seeing this error because you have `DEBUG = True` in your Django settings file. Change that to `False`, and Django will display a standard 404 page.

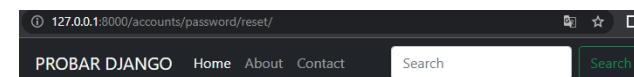
Pero si nos vamos a /account/logout/ vemos que la sesión se ha terminado.



Sesión terminada.

[Login](#)

Si el usuario quisiera cambiar de contraseña tendríamos que poner
"/accounts/password/reset"



Correo electrónico*

```
settings.py      activation_email.txt      password_reset_email.html      urls.py
{% load i18n %}
{% blocktrans %}Restablecer contraseña en {{ site_name }}{% endblocktrans %}:
{% block reset_link %}
{{ protocol }}://{{ domain }}{% url
'auth_password_reset_confirm' uid token %}
{% endblock %}
```

32. Cambiar URL Redirect después de login

Al intentar iniciar sesión nos da un erro

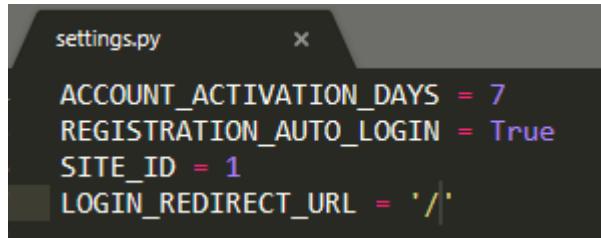
The screenshot shows a 'Page not found (404)' error page from a Django application. It displays the following information:

- Request Method: GET
- Request URL: http://127.0.0.1:8000/accounts/profile/
- Using the URLconf defined in proyecto.urls. Django tried these URL patterns, in this order:

 1. admin/
 2. [name='inicio']
 3. contact/ [name='contact']
 4. about/ [name='about']
 5. accounts/ activate/complete/ [name='registration_activation_complete']
 6. accounts/ activate/resend/ [name='registration_resend_activation']
 7. accounts/ activate/<activation_key>/ [name='registration_activate']
 8. accounts/ register/complete/ [name='registration_complete']
 9. accounts/ register/closed/ [name='registration_disallowed']
 10. accounts/ register/ [name='registration_register']
 11. accounts/ login/ [name='auth_login']
 12. accounts/ logout/ [name='auth_logout']
 13. accounts/ password/change/ [name='auth_password_change']
 14. accounts/ password/change/done/ [name='auth_password_change_done']
 15. accounts/ password/reset/ [name='auth_password_reset']
 16. accounts/ password/reset/complete/ [name='auth_password_reset_complete']
 17. accounts/ password/reset/done/ [name='auth_password_reset_done']
 18. accounts/ password/reset/confirm/uidb64/<token>/ [name='auth_password_reset_confirm']
 19. ^static/(?:path|.*)\$
 20. ^media/(?:path|.*)\$

- The current path, accounts/profile/, didn't match any of these.
- You're seeing this error because you have DEBUG = True in your Django settings file. Change that to False, and Django will display a standard 404 page.

Vamos a cambiar esta página, para configurar un redirect, para que nos lleve a otra.
Abrimos nuestro 'settings.py' y escribimos esto:



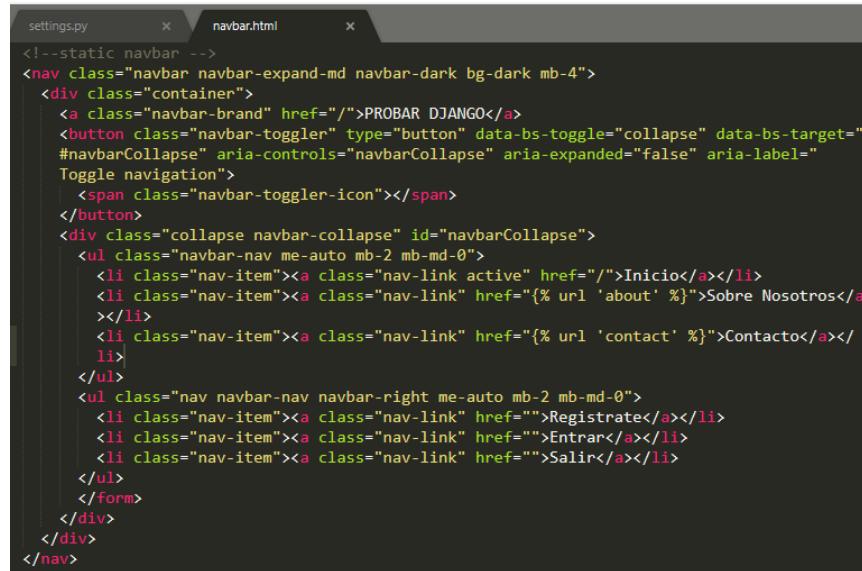
Con la raíz, decimos que una vez iniciada la sesión, nos vuelva a la página inicial nuestra.

The screenshot shows the 'Probar Django' website. The navigation bar at the top includes links for Home, About, Contact, and a search bar. The main content area features the title 'Probar Django' and a brief description: 'Un proyecto para principiantes. El objetivo es construir una página web simple a la par que elegante en muy poco tiempo'. Below this is a blue button labeled 'Únete >'. The right side of the page is a large black rectangular area. The footer contains a registration form with fields for Nombre and Email*, and a 'Regístrate' button. The footer text also includes: 'Bienvenido Fran', 'Creado con Django & Bootstrap', 'Y con mucho amor, claro', and 'Código abierto, siempre'.

33. Autenticación para enlaces en la Navbar

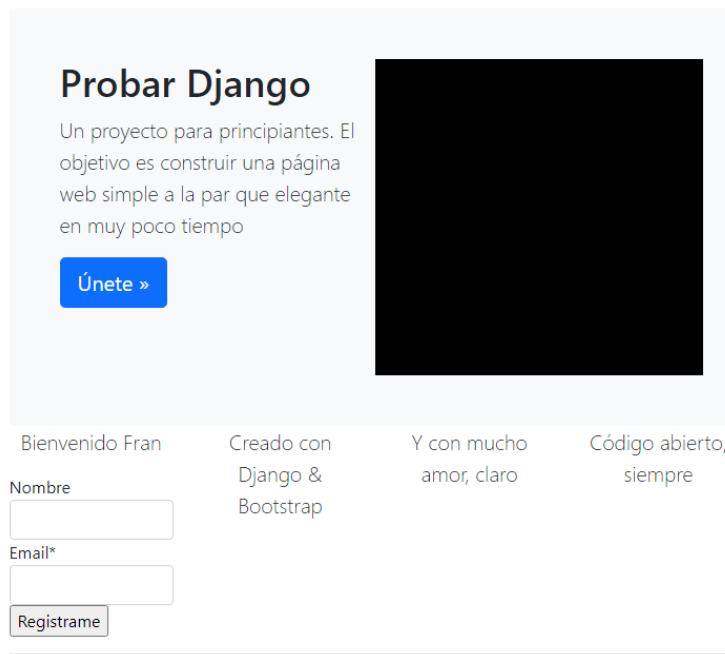
Ahora toca arreglar la barra de navegación para que también funcionen los enlaces. Abrimos 'navbar.html' y cambiamos todo al español, para que quede más presencial.

Para que al pinchar en algunas de las opciones de la barra nos redirige a un enlace tenemos que escribir lo siguiente:



```
<!--static navbar -->
<nav class="navbar navbar-expand-md navbar-dark bg-dark mb-4">
  <div class="container">
    <a class="navbar-brand" href="/">PROBAR DJANGO</a>
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarCollapse" aria-controls="navbarCollapse" aria-expanded="false" aria-label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarCollapse">
      <ul class="navbar-nav me-auto mb-2 mb-md-0">
        <li class="nav-item"><a class="nav-link active" href="/">Inicio</a></li>
        <li class="nav-item"><a class="nav-link" href="{% url 'about' %}">Sobre Nosotros</a></li>
        <li class="nav-item"><a class="nav-link" href="{% url 'contact' %}">Contacto</a></li>
      </ul>
      <ul class="nav navbar-nav navbar-right me-auto mb-2 mb-md-0">
        <li class="nav-item"><a class="nav-link" href="">Registrate</a></li>
        <li class="nav-item"><a class="nav-link" href="">Entrar</a></li>
        <li class="nav-item"><a class="nav-link" href="">Salir</a></li>
      </ul>
    </div>
  </div>
</nav>
```

una vez estemos logeados, tendremos un botón para poder salir de la sesión.



Probar Django

Un proyecto para principiantes. El objetivo es construir una página web simple a la par que elegante en muy poco tiempo

Únete >

Bienvenido Fran

Creado con Django & Bootstrap

Y con mucho amor, claro

Código abierto, siempre

Modificamos el fichero navbar.html para que al iniciar sesión podamos salir de la sesión.

```

settings.py      navbar.html

<nav class="navbar navbar-expand-md navbar-dark bg-dark mb-4">
  <div class="container">
    <a class="navbar-brand" href="/">PROBAR DJANGO</a>
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarCollapse" aria-controls="navbarCollapse" aria-expanded="false" aria-label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarCollapse">
      <ul class="navbar-nav me-auto mb-2 mb-md-0">
        <li class="nav-item"><a class="nav-link active" href="/">Inicio</a></li>
        <li class="nav-item"><a class="nav-link" href="{% url 'about' %}">Sobre Nosotros</a></li>
        <li class="nav-item"><a class="nav-link" href="{% url 'contact' %}">Contacto</a></li>
      </ul>
      <ul class="nav navbar-nav navbar-right me-auto mb-2 mb-md-0">
        {% if request.user.is_authenticated %}
          <li class="nav-item"><a class="nav-link" href="">Salir</a></li>
        {% else %}
          <li class="nav-item"><a class="nav-link" href="">Registrate</a></li>
          <li class="nav-item"><a class="nav-link" href="">Entrar</a></li>
        {% endif %}
      </ul>
    </div>
  </div>
</nav>

```

Al estar con la sesión iniciada solo nos aparecerá Salir

PROBAR DJANGO Inicio Sobre Nosotros Contacto Salir

Probar Django

Un proyecto para principiantes. El objetivo es construir una página web simple a la par que elegante en muy poco tiempo

[Únete »](#)

Bienvenido Fran	Creado con Django & Bootstrap	Y con mucho amor, claro	Código abierto, siempre
Nombre <input type="text"/>			
Email* <input type="text"/>			
<input type="button" value="Regístrate"/>			

Para configurar los botones tendremos que poner lo siguiente en el fichero "navbar.html":

```

settings.py
navbar.html

<!--static navbar -->
<nav class="navbar navbar-expand-md navbar-dark bg-dark mb-4">
  <div class="container">
    <a class="navbar-brand" href="/">PROBAR DJANGO</a>
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarCollapse" aria-controls="navbarCollapse" aria-expanded="false" aria-label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarCollapse">
      <ul class="navbar-nav me-auto mb-2 mb-md-0">
        <li class="nav-item"><a class="nav-link active" href="/">Inicio</a></li>
        <li class="nav-item"><a class="nav-link" href="{% url 'about' %}">Sobre Nosotros</a></li>
        <li class="nav-item"><a class="nav-link" href="{% url 'contact' %}">Contacto</a></li>
      </ul>
      <ul class="nav navbar-nav navbar-right me-auto mb-2 mb-md-0">
        {% if request.user.is_authenticated %}
          <li class="nav-item"><a class="nav-link" href="{% url 'auth_logout' %}">Salir</a></li>
        {% else %}
          <li class="nav-item"><a class="nav-link" href="{% url 'registration_register' %}">Registrate</a></li>
          <li class="nav-item"><a class="nav-link" href="{% url 'auth_login' %}">Entrar</a></li>
        {% endif %}
      </ul>
    </div>
  </div>
</nav>

```

34. Formulario de login en la Navbar.

Para hacer sesión hay que pinchar en 'Entrar', pero nosotros queremos tener en la página inicial un formulario para poder iniciar sesión.

Vamos a 'navbar.html' y creamos nuestro propio formulario:

```

settings.py
navbar.html

<!--static navbar -->
<nav class="navbar navbar-expand-md navbar-dark bg-dark mb-4">
  <div class="container">
    <a class="navbar-brand" href="/">PROBAR DJANGO</a>
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarCollapse" aria-controls="navbarCollapse" aria-expanded="false" aria-label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarCollapse">
      <ul class="navbar-nav me-auto mb-2 mb-md-0">
        <li class="nav-item"><a class="nav-link active" href="/">Inicio</a></li>
        <li class="nav-item"><a class="nav-link" href="{% url 'about' %}">Sobre Nosotros</a></li>
        <li class="nav-item"><a class="nav-link" href="{% url 'contact' %}">Contacto</a></li>
      </ul>
      <form class="navbar-form navbar-right" method="POST" action="{% csrf_token %}">
        <div class="from-group">
          <input type="text" class="form-control" name="username" placeholder="Usuario"/>
        </div>
        <div class="from-group">
          <input type="password" class="form-control" name="password" placeholder="Contraseña"/>
        </div>
        <button type="submit" class="btn btn-default">Entrar</button>
      </form>
      <ul class="nav navbar-nav navbar-right me-auto mb-2 mb-md-0">
        {% if request.user.is_authenticated %}
          <li class="nav-item"><a class="nav-link" href="{% url 'auth_logout' %}">Salir</a></li>
        {% else %}
          <li class="nav-item"><a class="nav-link" href="{% url 'registration_register' %}">Registrate</a></li>
          <li class="nav-item"><a class="nav-link" href="{% url 'auth_login' %}">Entrar</a></li>
        {% endif %}
      </ul>
    </div>
  </div>
</nav>

```



Probar Django

Un proyecto para principiantes. El objetivo es construir una página web simple a la par que elegante en muy poco tiempo

[Únete »](#)

Bienvenido

Creado con Django & Bootstrap

Y con mucho amor, claro

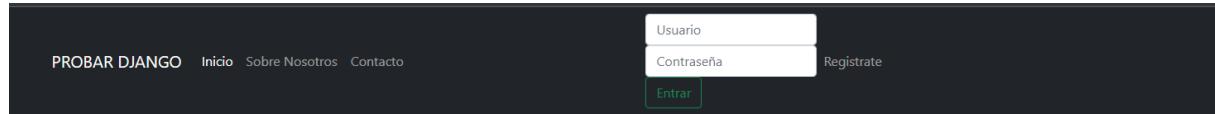
Código abierto, siempre

Nombre

Ahora vamos a modificar el fichero "navbar.html" para que solo nos salga un Entrar

A screenshot of a code editor showing the 'navbar.html' template file. The file contains HTML and Django templating code. It includes a login form with fields for 'username' and 'password', and a 'Entrar' button. Below the form, it shows a navigation bar with items for 'Salir' and 'Registrarse'. A conditional block checks if the user is authenticated, showing the 'Salir' link if true and the 'Registrarse' link if false. The code is syntax-highlighted with colors for different tags and variables.

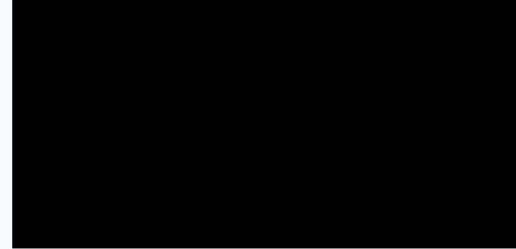
Veremos lo siguiente



Probar Django

Un proyecto para principiantes. El objetivo es construir una página web simple a la par que elegante en muy poco tiempo

[Únete »](#)



Bienvenido

Creado con Django & Bootstrap

Y con mucho amor, claro

Código abierto, siempre

Nombre

Para que al estar logeado no nos salga el botón Entrar pondremos lo siguiente.

```
Toggle navigation">
    <span class="navbar-toggler-icon"></span>
</button>
<div class="collapse navbar-collapse" id="navbarCollapse">
    <ul class="navbar-nav me-auto mb-2 mb-md-0">
        <li class="nav-item"><a class="nav-link active" href="/">Inicio</a></li>
        <li class="nav-item"><a class="nav-link" href="{% url 'about' %}">Sobre Nosotros</a></li>
        <li class="nav-item"><a class="nav-link" href="{% url 'contact' %}">Contacto</a></li>
    </ul>

    <ul class="nav navbar-nav navbar-right me-auto mb-2 mb-md-0">
        {% if request.user.is_authenticated %}
            <li class="nav-item"><a class="nav-link" href="{% url 'auth_logout' %}">Salir</a></li>
        {% else %}
            <li class="nav-item"><a class="nav-link" href="{% url 'registration_register' %}">Registrate</a></li>
        {% endif %}
    </ul>
    {% if not request.user.is_authenticated %}
        <form class="navbar-form navbar-right" method="POST" action="">{% csrf_token %}
            <div class="from-group">
                <input type="text" class="form-control" name="username" placeholder="Usuario"/>
            </div>
            <div class="from-group">
                <input type="text" class="form-control" name="password" placeholder="Contraseña"/>
            </div>
            <button type="submit" class="btn btn-outline-success">Entrar</button>
        </form>
    {% endif %}
    </div>
</div>
```

Resultado

Probar Django

Un proyecto para principiantes. El objetivo es construir una página web simple a la par que elegante en muy poco tiempo

[Únete »](#)



Bienvenido Fran

Nombre

Email*

[Regístrate](#)

Creado con

Django &
Bootstrap

Y con mucho

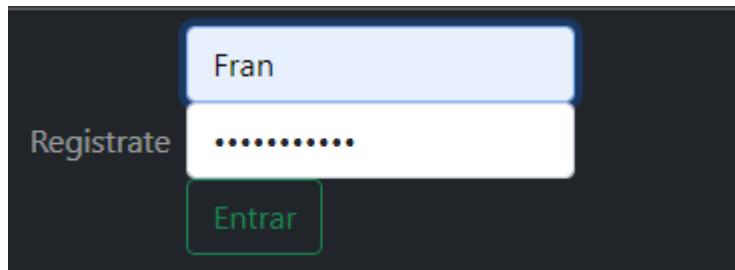
amor, claro

Código abierto,

siempre

volvemos a modificar el navbar.html para que la contraseña sea de tipo password para proteger la contraseña.

```
settings.py      navbar.html
  x          x
<form class="navbar-form navbar-right" method="POST" action="{% url 'auth_login' %}>
  {% csrf_token %}
  <div class="from-group">
    <input type="text" class="form-control" name="username" placeholder="Usuario"/>
  </div>
  <div class="from-group">
    <input type="password" class="form-control" name="password" placeholder="Contraseña"/>
  </div>
  <button type="submit" class="btn btn-outline-success">Entrar</button>
</form>
  {% endif %}
</form>
</div>
</nav>
```



Modificamos el fichero para que al iniciar sesión no salga el cuadrado de arriba para logearnos.

```
settings.py      x      navbar.html      x
...
    {% endif %}
  </ul>
  {% if not request.user.is_authenticated and not "/accounts/login" in
request.get_full_path %}
    <form class="navbar-form navbar-right" method="POST" action="{% url 'auth_login' %}>
      {% csrf_token %}
      <div class="from-group">
        <input type="text" class="form-control" name="username" placeholder="Usuario"/>
      </div>
      <div class="from-group">
        <input type="password" class="form-control" name="password" placeholder="Contraseña"/>
      </div>
      <button type="submit" class="btn btn-outline-success">Entrar</button>
    </form>
    {% endif %}
  </div>
</div>
</nav>
```



Iniciar Sesión

Nombre de usuario*

Contraseña*

Iniciar sesion

Has olvidado tu contraseña? [Restablecer!](#)

No tienes cuenta? [Registrarte!](#)